ProvIDe - Bank-Verified Digital Identity Platform

Final Project Definition

By:

- Amit Zamir (ID: 322833021, Email: Im.amit.zamir@gmail.com)
- Noa Almakais (ID: 211632302, Email: noaelmakies1@gmail.com)
- May Gorvitz (ID: 322438177, Email: mht1445@gmail.com)
- Hai Tal (ID: 311561245, Email: haital18@gmail.com)

Supervisor: Ari Ben Ephraim

Git: https://github.com/Amit-Za/provide.git

Table of Contents

1. Project Description

ProvIDe is a next-generation digital identity infrastructure that revolutionizes how third-party platforms verify user identity. Instead of requiring users to upload identity documents—a process that introduces friction, security risks, and regulatory complexity—ProvIDe leverages the user's existing bank relationship as a trusted identity anchor.

The platform securely delegates authentication to regulated banks via OpenFinance.ai and returns cryptographically signed identity assertions that meet enterprise security and compliance standards. The system can recieve bank confirmations in real-time. This approach eliminates the need for document storage while providing bank-level security assurance.

The Problem

Current identity verification processes face several significant challenges:

- **High Friction:** Document-based KYC causes 30-40% user abandonment rates, creating barriers to service adoption
- **Security Risk:** Storing identity documents creates breach liability and regulatory compliance burdens
- **Redundant Verification:** Users must verify their identity repeatedly across different services, even though banks have already performed strict verification
- **Insufficient Alternatives:** Social login solutions lack legal assurance required for regulated use cases such as financial services

The Solution

ProvIDe bridges the gap between bank-level identity verification and third-party service needs, enabling instant, secure identity verification without document storage. The platform operates in two modes:

- **Identity Verification Mode:** One-time or ongoing verification with cryptographically signed assertions that can be used according to service policies
- **OAuth IdP Mode:** Full OAuth 2.0 / OpenID Connect provider implementation for seamless third-party login integration

Market Positioning

Current Market Gaps:

- **Open Banking:** Provides account data access but does not serve as an identity verification service
- **KYC Vendors:** Focus primarily on document processing rather than leveraging existing bank verification infrastructure
- **Social Login:** Convenient but lacks the legal assurance necessary for regulated services requiring verified identity

ProvIDe Advantage:

- Leverages existing bank KYC infrastructure that has already performed strict verification
- No document storage or processing required, reducing security risks and compliance complexity
- Bank-level security with enterprise compliance standards built-in

- Standard OAuth 2.0 / OpenID Connect integration for easy third-party adoption

The platform integrates securely with banking infrastructure through OpenFinance.ai, enabling seamless connectivity with regulated banking systems while maintaining strict security and compliance standards.

2. Related Work

Today, digital services that require identity verification rely primarily on classic KYC processes, which include uploading identification documents, taking selfies, and manual or automatic verification. These processes are characterized by high friction, complex user experience, and significant risks related to storing sensitive documents and fraud prevention.

In parallel, there are Open Banking and Open Finance solutions that allow access to the user's financial data, but these solutions do not serve as identity providers and do not provide official confirmation of the user's identity to third parties. Banks, despite being the entity that performed the strictest identification, do not currently serve as digital identity providers for external services.

As a result, many organizations continue to require uploading physical documents, even when it would be possible to rely on an identity that has already been verified at the banking level.

Current Solutions & Limitations

Traditional KYC Providers: Rely on document upload and biometrics, introducing high friction and security risks. These solutions require users to submit sensitive documents that must be stored and processed, creating liability and compliance challenges.

Open Banking Platforms: Expose financial data but do not provide identity assurance or verification services. While they enable access to account information, they cannot confirm identity attributes to third parties.

Global Identity Providers (Social Login): Convenient but not suitable for regulated use cases requiring legal assurance. Services like Google and Facebook authentication lack the regulatory compliance needed for financial services and other highly regulated industries.

Bank Identity Gap: Banks perform the strictest verification during account opening but don't currently serve as identity providers for external services, creating a significant market opportunity.

Technical Foundation

The platform is built on industry-standard protocols and technologies:

- **OAuth 2.0:** Authorization framework for secure authentication flows, providing the foundation for delegated access
- **OpenID Connect:** Identity layer on top of OAuth 2.0 for identity verification, enabling standardized identity claims
- **JWT (JSON Web Token):** Cryptographically signed identity assertions that can be verified independently
- **JWS (JSON Web Signature):** Digital signatures for token integrity, ensuring assertions cannot be tampered with

These standards ensure interoperability, security, and compliance with industry best practices while enabling seamless integration with existing third-party systems.

3. Functional Description / Requirements

General System Description

Third-party platforms integrate a standard login or verification flow into their services. Users authenticate via their bank using Strong Customer Authentication (SCA) methods. The platform receives confirmation and issues a signed identity token containing only the claims the platform is authorized to receive.

A digital service integrates an authentication button—"Identity Verification via Bank" or "Login / Verify with Bank"—into the user interface. After clicking the button, the user is redirected to an authentication process with their bank through a secure redirect via OpenFinance.ai.

During authentication, the bank performs strong verification using existing means (password, OTP, biometrics), and after user approval, returns a digital identity confirmation (Identity Assertion) to the system. This confirmation includes verified identity details as needed, such as name match, ID number, age, or country, without transferring documents or financial data.

The system provides the digital service with a reliable indication of the user's identity, which can be used one-time or continuously, according to the service's policy.

Core Use Cases

Loan & Financial Services: Eliminate document uploads and selfie verification. Enable instant account creation with bank-verified identity for loans, credit applications, and financial products. This reduces abandonment rates and accelerates customer onboarding.

E-commerce & Trading: Verify customers and sellers instantly. Reduce identity fraud, enable verified account creation, and streamline onboarding for trading platforms and marketplaces. This is particularly valuable for platforms requiring age verification or seller authentication.

Regulated Services: Insurance, cryptocurrency platforms, age-restricted services, and highly regulated industries requiring verified identity with legal assurance. ProvIDe provides the compliance-grade verification needed for these sectors.

Detailed Functional Requirements

- **Third-party platform registration and approval:** Services must register and be approved before accessing the platform, ensuring only legitimate services can use the identity verification system
- **OAuth authorization and token issuance:** Secure OAuth 2.0 Authorization Code Flow for bank authentication, following industry standards
- **Explicit user consent with clear scope definition:** Users must explicitly consent to attribute sharing with clear visibility of what is being shared, ensuring transparency and GDPR compliance
- **Policy-based attribute disclosure:** Configurable identity attributes (name, ID, age, country) based on service needs and user consent, allowing fine-grained control
- **Time-limited and revocable identity assertions:** Identity tokens have expiration and can be revoked by users, providing ongoing control
- **Identity assertion generation and validation:** Cryptographically signed JWT tokens containing verified identity claims, ensuring integrity and authenticity
- **Token management and lifecycle handling:** Secure storage, refresh, and revocation of tokens, maintaining security throughout the token lifecycle
- **Audit logging for compliance and security:** Comprehensive logging of all identity verification events, enabling compliance reporting and security monitoring
- **OpenFinance.ai integration:** Bank connectivity via OpenFinance.ai for secure banking infrastructure access, leveraging existing open banking infrastructure
- **RTL (Right-to-Left) UI support:** Hebrew interface support for consent screens and user interactions, ensuring accessibility for Hebrew-speaking users
- **Responsive design:** Mobile and desktop device optimization, ensuring the platform works across all devices
- **No storage of sensitive data:** No storage of identification documents or bank login credentials, minimizing security risks

- **Dual mode operation:** Support for both identity verification mode and Identity Provider (IdP) mode, providing flexibility to accomodate different use cases

4. Architecture

The system architecture consists of three main components: the client-side application, teh server-side platform, and the bank authentication layer. The architecture follows a microservices approach with clear seperation of concerns, ensuring scalability, maintainability, and security.

Built on industry-standard protocols with a microservices architecture designed for security, scalability, and regulatory compliance. The system integrates with OpenFinance.ai for bank connectivity and implements OAuth 2.0 / OpenID Connect standards.

System Architecture Diagram

The system follows this flow:

1. User requests authentication from Client App
2. Client App sends OAuth request to ProvIDe Platform
3. ProvIDe redirects to Bank for identity verification
4. Bank verifies identity and confirms to ProvIDe
5. ProvIDe returns authentication token to Client App
6. Client App uses token to access Digital Services

System Components

Client Side:

- Angular or React framework for modern web development
- RTL UI support for Hebrew interface
- OAuth redirect handling for secure authentication flows
- Responsive design for mobile and desktop devices

Server Side:

- Node.js with Express.js for the runtime and web framework
- PostgreSQL with Prisma ORM for data persistence
- Redis for caching, token blacklisting, and rate limiting
- RabbitMQ for asynchronous inter-service communication
- REST APIs for service integration
- Identity management and token handling

- Token and permissions management

Bank Authentication:

- OAuth 2.0 Authorization Code Flow for secure bank authentication
- Open Banking intermediaries for bank connectivity
- Secure bank connections via OpenFinance.ai

Security & Compliance:

- Data encryption at rest and in transit
- Token management with secure storage and lifecycle handling
- Audit mechanisms for compliance and security monitoring
- Regulatory compliance with GDPR and financial services regulations

Microservices Architecture

The server-side platform is built on a microservices architecture, ensuring scalability, security, and regulatory compliance. Microservices communicate via REST APIs for synchronous operations and RabbitMQ for asynchronous event-driven processing.

Inter-Service Communication:

Microservices communicate through a combination of synchronous HTTP/REST APIs and asynchronous message queues using RabbitMQ for event-driven architecture.

RabbitMQ Use Cases:

- Audit log event processing (async)
- Token revocation broadcasts
- Bank callback processing
- Email/notification queues
- Background job processing
- Service-to-service event notifications

Direct HTTP Use Cases:

- OAuth token exchange (synchronous)
- Service registration (immediate validation)
- Real-time API calls requiring immediate response
- User consent screen interactions

RabbitMQ Architecture:

Message Queues & Exchanges:

- `audit.events` - Audit log events queue
- `token.revocation` - Token revocation broadcasts
- `bank.callbacks` - Bank OAuth callback processing
- `notifications` - Email/notification queue
- `consent.events` - Consent lifecycle events

Communication Patterns:

- **Pub/Sub:** Token revocation events broadcast to all services
- **Work Queues:** Audit log processing with multiple workers
- **Routing:** Bank-specific callback routing
- **Dead Letter Queues:** Failed message handling and retry logic

Benefits:

- Decoupled microservices (services don't need to know about each other)
- Reliable message delivery with acknowledgments
- Automatic retry and dead-letter queue handling
- Scalable processing with multiple consumers
- Message persistence for audit and recovery

Database Schema & Models

users Table:

- `id` (UUID, PRIMARY KEY): Unique user identifier
- `bank_user_id` (VARCHAR(255), UNIQUE, NOT NULL): Bank-provided user identifier (hashed)
- `bank_id` (INTEGER, FOREIGN KEY, NOT NULL): Reference to banks table
- `verification_status` (ENUM, NOT NULL, DEFAULT 'pending'): pending, verified, expired, revoked
- `verified_at` (TIMESTAMP, NULL): Last verification timestamp
- `created_at` (TIMESTAMP, NOT NULL, DEFAULT NOW()): Account creation timestamp
- `updated_at` (TIMESTAMP, NOT NULL, DEFAULT NOW()): Last update timestamp

Note: No PII stored. Only verification status and bank reference.

services Table:

- `id` (UUID, PRIMARY KEY): Unique service identifier
- `name` (VARCHAR(255), NOT NULL): Service display name
- `client_id` (VARCHAR(255), UNIQUE, NOT NULL): OAuth client identifier
- `client_secret_hash` (VARCHAR(255), NOT NULL): Bcrypt hashed client secret
- `redirect_uris` (JSONB, NOT NULL): Array of allowed redirect URIs
- `status` (ENUM, NOT NULL, DEFAULT 'pending'): pending, approved, suspended, revoked
- `api_key` (VARCHAR(255), UNIQUE): API key for service authentication
- `created_at` (TIMESTAMP, NOT NULL, DEFAULT NOW()): Registration timestamp

authorizations Table:

- `id` (UUID, PRIMARY KEY): Unique authorization record ID
- `user_id` (UUID, FOREIGN KEY, NOT NULL): Reference to users table
- `service_id` (UUID, FOREIGN KEY, NOT NULL): Reference to services table
- `scopes` (JSONB, NOT NULL): Granted OAuth scopes (e.g., ["name", "age"])
- `consent_given_at` (TIMESTAMP, NOT NULL): User consent timestamp
- `consent_revoked_at` (TIMESTAMP, NULL): Consent revocation timestamp
- `status` (ENUM, NOT NULL, DEFAULT 'active'): active, revoked, expired

tokens Table:

- `id` (UUID, PRIMARY KEY): Unique token record ID
- `authorization_id` (UUID, FOREIGN KEY, NOT NULL): Reference to authorizations table
- `token_type` (ENUM, NOT NULL): access_token, refresh_token, id_token
- `token_hash` (VARCHAR(255), UNIQUE, NOT NULL): SHA-256 hash of JWT (for revocation)
- `expires_at` (TIMESTAMP, NOT NULL): Token expiration timestamp
- `revoked_at` (TIMESTAMP, NULL): Revocation timestamp
- `created_at` (TIMESTAMP, NOT NULL, DEFAULT NOW()): Token creation timestamp

Security: Only token hashes stored. Actual JWTs are stateless and validated via signature.

audit_logs Table:

- `id` (BIGSERIAL, PRIMARY KEY): Sequential log entry ID
- `event_type` (ENUM, NOT NULL): auth_request, consent_given, token_issued, token_revoked, assertion_issued
- `user_id` (UUID, FOREIGN KEY, NULL): User involved (if applicable)
- `service_id` (UUID, FOREIGN KEY, NULL): Service involved (if applicable)

- `ip_address` (INET, NULL): Client IP address
- `user_agent` (TEXT, NULL): Client user agent
- `metadata` (JSONB, NULL): Event-specific metadata
- `created_at` (TIMESTAMP, NOT NULL, DEFAULT NOW()): Event timestamp

Retention: GDPR-compliant retention policy. Immutable logs with tamper detection.

banks Table:

- `id` (INTEGER, PRIMARY KEY): Bank identifier
- `name` (VARCHAR(255), NOT NULL, UNIQUE): Bank display name
- `openfinance_bank_id` (VARCHAR(255), UNIQUE, NOT NULL): OpenFinance.ai bank identifier
- `oauth_endpoint` (VARCHAR(500), NOT NULL): Bank OAuth authorization endpoint
- `token_endpoint` (VARCHAR(500), NOT NULL): Bank OAuth token endpoint
- `is_active` (BOOLEAN, NOT NULL, DEFAULT true): Bank integration status

Security Architecture Details

Encryption & Hashing:

- TLS 1.3: All external communications
- AES-256-GCM: Data encryption at rest
- Bcrypt (cost 12): Client secrets hashing
- SHA-256: Token hash for revocation tracking
- RSA-2048 / ECDSA-P256: JWT signing keys

Key Management:

- JWKS: Public key distribution endpoint
- Key Rotation: Automated key rotation every 90 days
- HSM Integration: Private keys stored in HSM (production)
- Key Versioning: Support for multiple active keys

Token Security:

- Short-lived Access Tokens: 15 minutes expiration
- Refresh Tokens: 30 days, single-use, revocable
- PKCE: Code challenge for public clients
- State Parameter: CSRF protection
- Nonce: Replay attack prevention

Access Control:

- Scope-based: Fine-grained permissions
- Rate Limiting: Per-client, per-endpoint
- IP Whitelisting: Optional for services
- CORS: Strict origin validation

Detailed Data Flow

1. Authorization Request Flow:

1. Client redirects user to `/oauth/authorize` with client_id, redirect_uri, scope, state, code_challenge
2. Authorization Server validates client_id, redirect_uri, generates authorization_code
3. Authorization Server stores code with expiration (10 min), PKCE challenge, scopes
4. User redirected to bank selection → OpenFinance.ai → Bank OAuth
5. Bank authenticates user, returns to OpenFinance.ai callback
6. OpenFinance.ai validates and forwards to ProvIDe callback
7. ProvIDe creates/updates user record, sets verification_status = 'verified'
8. User redirected to consent screen with requested scopes
9. User approves → authorization record created in database
10. User redirected back to client with authorization_code

2. Token Exchange Flow:

1. Client POSTs to `/oauth/token` with authorization_code, code_verifier, client_id, client_secret
2. Authorization Server validates code, verifies PKCE challenge
3. Authorization Server retrieves authorization record, validates expiration
4. Identity Assertion Engine generates JWT ID token with user claims (based on scopes)
5. Access token (JWT) generated with scopes, user_id, service_id
6. Refresh token generated, stored in database (hashed)
7. Token records created in tokens table (access_token, refresh_token, id_token)
8. Audit log entry created
9. Tokens returned to client (access_token, refresh_token, id_token, expires_in)

3. Identity Assertion Generation:

1. Identity Assertion Engine receives verified user data from bank
2. Scopes filtered based on authorization record
3. JWT header created: {"alg": "RS256", "typ": "JWT", "kid": "key-id"}
4. JWT payload created with: iss, sub, aud, exp, iat, jti, and scope-based claims
5. Claims include only authorized attributes (name, age, country, etc.)

6. JWT signed using RS256 with private key
7. JWS signature appended
8. ID token record created in tokens table (token_type = 'id_token')
9. JWT returned as id_token in token response

Technology Stack & Design Patterns

Backend Stack:

- Runtime: Node.js 18+
- Framework: Express.js
- Database: PostgreSQL 15+ (ACID, transactions)
- ORM: Prisma
- Caching: Redis (token blacklist, rate limiting, session storage)
- Message Queue: RabbitMQ (async processing, event-driven communication)

Design Patterns:

- Microservices: Service-oriented architecture
- Repository Pattern: Data access abstraction
- Factory Pattern: Service instantiation
- Strategy Pattern: Bank-specific adapters
- Observer Pattern: Event-driven audit logging
- Circuit Breaker: External API resilience

Security Libraries:

- JWT: jsonwebtoken / jose
- OAuth: node-oauth2-server / oauth2orize
- Crypto: Node crypto
- Hashing: bcrypt

Infrastructure:

- Containerization: Docker
- Load Balancing: NGINX
- Monitoring: Prometheus, Grafana (TBD)
- Logging: Loki (with Grafana)

API Design Patterns

RESTful Principles:

- Resource-based URLs (`/api/v1/identity/assertions/:id`)
- HTTP methods: GET (read), POST (create), PUT (update), DELETE (remove)
- Status codes: 200 (success), 201 (created), 400 (bad request), 401 (unauthorized), 404 (not found), 500 (server error)
- JSON request/response format

Error Handling:

- Consistent error format: `{"error": "error_code", "error_description": "human readable"}`
- Validation errors return 400 with field-level details
- Rate limiting returns 429 with Retry-After header
- All errors logged for debugging (without sensitive data)

4.1. Each Module Description

Authentication Module

Orchestrates OAuth 2.0 Authorization Code Flow, manages secure redirects, and handles token exchange. Integrates with bank authentication endpoints via OpenFinance.ai, ensuring secure communication through authorized Open Banking intermediaries.

The module handles the complete authorization flow from initial client request through bank authentication and token issuance. It validates client credentials, manages authorization codes, and ensures secure state management throughout the process.

Identity Assertion Service

Generates cryptographically signed JWT identity assertions from bank-verified user data. Manages policy-based attribute disclosure, ensures GDPR compliance, and validates assertion integrity using JWS signatures.

This service is responsible for creating identity tokens that contain only the attributes authorized by user consent and service requirements. It ensures that sensitive data is properly filtered before being included in tokens. It implements strict filtering to ensure minimal data disclosure while maintaining verification integrity.

Token Management

Securely manages OAuth access tokens, refresh tokens, and session state. Implements encrypted storage, automatic expiration handling, and user-initiated revocation mechanisms for enhanced security.

The token management module handles the complete lifecycle of authentication tokens, from generation through expiration and revocation. It ensures tokens are properly stored, validated, and can be revoked when necessary for security purposes.

API Gateway

Provides enterprise-grade RESTful API endpoints for service integration. Implements request validation, rate limiting, API versioning, and comprehensive error handling for production reliability.

The API Gateway serves as the entry point for all external service interactions, ensuring consistent security policies, request validation, and proper error handling across all endpoints.

Audit & Compliance

Comprehensive event logging and audit trail maintenance for regulatory compliance. Implements GDPR-compliant data retention policies and ensures adherence to financial services regulations.

This module captures all significant events in the system, including authentication requests, consent decisions, token issuance, and revocation events. It provides the audit trail necessary for regulatory compliance and security monitoring.

Client UI Module

Provides embeddable authentication components, secure redirect handling, and real-time user feedback. Features full RTL support for Hebrew interface and responsive design optimized for all device types.

The UI module ensures users have a clear, accessible interface for consent and authentication processes. It handles the user-facing aspects of the identity verification flow, including bank selection, consent screens, and status feedback.

5. Work Plan

Team Members

- **Amit Zamir** (ID: 322833021, Email: lm.amit.zamir@gmail.com)
- **Noa Almakais** (ID: 211632302, Email: noaelmakies1@gmail.com)
- **May Gorvitz** (ID: 322438177, Email: mht1445@gmail.com)
- **Hai Tal** (ID: 311561245, Email: haital18@gmail.com)

Project Milestones

Milestone 1: Regulatory Onboarding & OpenFinance.ai Integration

Regulatory compliance setup, OpenFinance.ai platform integration, and secure bank connectivity implementation. This milestone establishes the foundation for bank communication and ensures compliance with financial regulations.

Responsibility: Backend team members

Milestone 2: Consent Orchestration & Assertion Engine

Implementation of consent orchestration service, identity assertion engine with JWT generation, and policy-based attribute disclosure. This milestone creates the core identity verification functionality.

Responsibility: Backend team members

Milestone 3: OAuth IdP Enablement & Client Registry

OAuth 2.0 / OpenID Connect Identity Provider implementation, client registry service, and third-party platform registration system. This milestone enables third-party service integration.

Responsibility: Backend team members

Milestone 4: Frontend Development

Client-side application with authentication UI, consent screens, redirect handling, RTL support, and responsive design. This milestone creates the user-facing components of the platform.

Responsibility: Frontend team members

Milestone 5: Security Hardening, Auditing & Documentation

Security hardening, comprehensive audit logging, compliance verification, end-to-end testing, and final documentation. This milestone ensures production readiness and regulatory compliance.

Responsibility: All team members

Note: Detailed work plan with specific tasks and timelines will be managed using team management software (Monday, Trello, or Jira).

6. Client Side

The client-side application provides a seamless user experience for bank-verified identity authentication. Built with modern web frameworks (Angular or React), the application features a clean, intuitive interface with RTL support for Hebrew users.

Key Features

- Unified authentication button component for easy integration into third-party services
- Secure OAuth redirect handling and callback management
- RTL (Right-to-Left) UI support for Hebrew interface
- Responsive design optimized for mobile and desktop devices
- Real-time authentication status feedback
- Error handling and user-friendly error messages
- Dark mode support for better user experience

The client application serves as the user interface for the identity verification process, handling all user interactions from initial authentication request through bank login and consent approval.

6.1. Usage Illustration

Scenario 1: Fintech Onboarding

1. User visits a fintech service website and initiates account creation
2. Service displays "Login / Verify with Bank" button
3. User clicks the button and is redirected to consent screen showing requested attributes
4. User reviews and approves consent, then authenticates with bank via OpenFinance.ai
5. User authenticates with bank using Strong Customer Authentication (password, OTP, biometrics)

6. Bank confirms identity and redirects back to fintech service
7. Fintech service receives cryptographically signed identity assertion
8. Account is created instantly without document upload - identity is verified via bank

This scenario demonstrates how ProvIDe eliminates the friction of traditional document-based onboarding while maintaining security and compliance.

Scenario 2: Trading Account Activation

1. User wants to activate a trading account that requires age and residency verification
2. Trading platform requires identity verification for regulatory compliance
3. User clicks "Verify via Bank" authentication button
4. Consent screen displays requested attributes (age, country, name)
5. After bank authentication and user consent, platform receives verified identity assertion
6. Trading account is activated with verified status
7. User can start trading immediately without document verification delays

This scenario shows how ProvIDe enables instant verification for regulated services, reducing time-to-value for both users and service providers.

Scenario 3: Marketplace Seller Verification

1. User wants to create a seller account on a marketplace platform
2. Platform requires identity verification for sellers to prevent fraud
3. User authenticates via bank using ProvIDe platform
4. Consent screen shows what information will be shared (name, ID verification status)
5. Service receives verified identity assertion without full ID details
6. Seller account is created instantly with verified status
7. Privacy is maintained - only necessary information is shared

This scenario illustrates how ProvIDe enables privacy-preserving identity verification, sharing only the minimum necessary information while providing the assurance needed for fraud prevention.

6.2. Mockup

The client-side application includes several key screens and components:

Authentication Button Component: A unified button that can be embedded into third-party services, triggering the identity verification flow. The button displays "Verify Identity via Bank" and includes the ProvIDe branding.

Bank Selection Interface: After clicking the authentication button, users are presented with a list of supported banks. Users select their bank, which initiates the secure redirect to the bank's authentication system.

Bank Login Screen: Users authenticate with their bank using existing bank credentials and Strong Customer Authentication methods (password, OTP, biometrics). This screen is hosted by the bank, ensuring users trust the authentication process.

Consent Screen - Attribute Disclosure: After successful bank authentication, users see a clear consent screen showing which service is requesting access and what specific identity attributes will be shared. Users can review and approve or cancel the request.

Redirect / Loading State: During the authentication process, users see a loading screen indicating that they are being securely connected to their bank. This provides clear feedback during the redirect process.

Success Confirmation Screen: After successful verification, users see a confirmation screen indicating that their identity has been verified. They can then proceed to use the third-party service.

Error Handling Screen: If authentication fails, users see a clear error message with options to try again or contact support. This ensures users understand what went wrong and how to proceed.

OTP / Two-Factor Authentication: For banks requiring additional authentication, users may see an OTP input screen. This is handled seamlessly within the flow.

Service Integration Example - Fintech Onboarding: Shows how the authentication button integrates into a typical fintech onboarding flow, appearing alongside traditional registration fields.

Multi-Step Progress Indicator: For longer authentication flows, a progress indicator shows users which step they are on (Bank Selection, Bank Login, Verification).

Verification Status Dashboard: Users can view their verification status and see which services they have connected, providing transparency and control over their identity sharing.

All mockups are designed with responsive layouts that work on both mobile and desktop devices, with full RTL support for Hebrew-speaking users.

7. Server Side

The server-side platform serves as the core identity verification service, managing authentication flows, identity assertions, and API endpoints for digital services. Built with Node.js, Express.js, PostgreSQL, Prisma, Redis, and RabbitMQ, the server ensures security, scalability, and regulatory compliance.

Server Architecture

The server-side platform is built on a microservices architecture, ensuring scalability, security, and regulatory compliance. The system integrates with OpenFinance.ai for bank connectivity and implements OAuth 2.0 / OpenID Connect standards. Microservices communicate via REST APIs for synchronous operations and RabbitMQ for asynchronous event-driven processing.

Key architectural features:

- RESTful API architecture for service integration
- Microservices-based design for scalability and maintainability
- RabbitMQ message queue for async inter-service communication
- Redis for caching, token blacklisting, and rate limiting
- PostgreSQL with Prisma ORM for data persistence
- Secure token management and storage
- Identity assertion generation and validation using JWT
- Integration with OpenFinance.ai for bank connectivity
- Comprehensive audit logging system for compliance
- Data encryption at rest and in transit
- Rate limiting and API security measures

Microservices & Modules

OAuth Authorization Server: Handles OAuth 2.0 and OpenID Connect flows, token issuance, and authorization management. This service implements the complete OAuth 2.0 Authorization Code Flow with PKCE support.

Client Registry Service: Manages third-party platform registration, approval, and client credentials. Services must register and be approved before they can use the platform, ensuring only legitimate services can access identity verification.

Consent Orchestration Service: Orchestrates user consent flows, scope management, and consent revocation. This service ensures users have clear visibility into what information is being shared and maintains consent records for compliance.

OpenFinance.ai Integration Layer: Handles secure communication with banks via OpenFinance.ai API. This layer manages bank selection, OAuth flow orchestration, and callback processing.

Identity Assertion Engine: Generates and validates cryptographically signed identity assertions (JWT). This service creates identity tokens containing only the attributes authorized by user consent and service requirements.

Audit & Compliance Module: Comprehensive logging, audit trails, and compliance reporting. Consumes audit events from RabbitMQ queues for async processing, ensuring all significant events are logged for regulatory compliance.

Technology Stack

Core Technologies:

- Runtime: Node.js 18+
- Framework: Express.js
- Database: PostgreSQL 15+
- ORM: Prisma
- Caching: Redis
- Message Queue: RabbitMQ

RabbitMQ Integration:

- Library: amqplib (Node.js RabbitMQ client)
- Patterns: Pub/Sub, Work Queues, Routing
- Queues: audit.events, token.revocation, bank.callbacks, notifications
- Features: Message acknowledgments, dead-letter queues, retry logic

7.1. API

OAuth 2.0 / OpenID Connect Endpoints

GET /oauth/authorize

OAuth 2.0 authorization endpoint - initiates authentication flow. Accepts standard OAuth 2.0 parameters including client_id, redirect_uri, scope, state, and code_challenge (for PKCE).

POST /oauth/token

Token endpoint - exchanges authorization code for access/ID tokens. Validates authorization code, verifies PKCE challenge, and returns access token, refresh token, and ID token.

GET /.well-known/openid-configuration

OpenID Connect discovery endpoint - provides configuration metadata. Returns OpenID Connect discovery document with endpoints, supported features, and key information.

GET /keys/jwks.json

JSON Web Key Set endpoint - provides public keys for token verification. Services can use this endpoint to verify JWT signatures.

POST /v1/openfinance/oauth/callback

OpenFinance.ai OAuth callback handler - processes bank authentication responses. This endpoint receives callbacks from OpenFinance.ai after bank authentication completes.

Identity Verification Endpoints

POST /api/v1/identity/verify

Request identity verification for a user. Initiates the identity verification process for a specific user session.

GET /api/v1/identity/assertion/:id

Retrieve identity assertion by ID. Returns a specific identity assertion token for verification purposes.

GET /api/v1/identity/status/:sessionId

Check verification status. Returns the current status of an identity verification session.

Service Management Endpoints

POST /api/v1/services/register

Register a new digital service. Services use this endpoint to register and obtain client credentials for platform access.

GET /api/v1/services/:serviceId/config

Get service configuration and API keys. Returns service configuration including client credentials and API keys.

Audit & Compliance Endpoints

GET /api/v1/audit/logs

Retrieve audit logs (admin only). Returns audit log entries for compliance and security monitoring purposes.

GET /api/v1/compliance/report

Generate compliance reports. Creates compliance reports based on audit logs and system activity.

API Authentication

All API endpoints require authentication using API keys or OAuth 2.0 client credentials. Services must register and obtain API credentials before accessing the platform. The API implements rate limiting, request validation, and comprehensive error handling to ensure security and reliability.

8. References

Technologies & Standards

- **OAuth 2.0 (RFC 6749):** Authorization framework for secure authentication flows
- **OpenID Connect Core 1.0:** Identity layer on top of OAuth 2.0 for identity verification
- **JWT (RFC 7519):** JSON Web Token format for identity assertions
- **JWS (RFC 7515):** JSON Web Signature for cryptographically signed tokens
- **OpenFinance.ai:** https://docs.open-finance.ai/ - Open Banking platform for bank connectivity
- **REST API:** Architectural style for web services
- **Angular / React:** Frontend frameworks for client-side development
- **Node.js / Express.js:** Backend runtime and framework
- **PostgreSQL:** Relational database management system
- **Prisma:** Modern database ORM
- **Redis:** In-memory data structure store for caching
- **RabbitMQ:** Message broker for asynchronous communication

Regulatory & Compliance

- **KYC (Know Your Customer):** Customer identification requirements
- **GDPR:** European data protection regulation
- **Financial Regulation:** Banking and financial services compliance
- **PSD2:** Payment Services Directive (EU)

Related Products & Solutions

- **OpenFinance.ai:** Open Banking platform providing secure bank connectivity and API integration
- **Open Banking Platforms:** Existing financial data sharing solutions (focus on data, not identity)
- **Traditional KYC Providers:** Document-based identity verification solutions
- **Social Login Providers:** Google, Facebook authentication services (convenient but not suitable for regulated use cases)
- **Digital Identity Solutions:** eIDAS, national ID systems
- **Banking APIs:** Bank-provided API services

Security Standards

- **TLS/SSL:** Transport layer security for encrypted communication
- **PKI (Public Key Infrastructure):** Certificate-based security
- **Encryption:** Data protection at rest and in transit
- **Audit Logging:** Security event tracking and compliance