

Forward Inheritance SaaS Platform - Product Requirements Document V1.2

Executive Summary

Forward is building a family-first inheritance and wealth transfer SaaS platform centered around Forward Family Circles (FFCs) for collaborative estate planning and wealth transfer transparency, featuring integrated referral systems and comprehensive asset management capabilities.

Table of Contents

1. [Project Context](#)
2. [Strategic Insights](#)
3. [Phase 1A Scope & Core Features](#)
4. [Epic Structure](#)
5. [Technical Architecture](#)
6. [Database Schema](#)
7. [Success Metrics](#)
8. [Risk Mitigation](#)
9. [Development Phases](#)

Project Context

Forward is developing a SaaS platform that transforms traditional individual-focused wealth management into collaborative family-centered inheritance planning, providing comprehensive asset management and estate planning collaboration.

Vision

Create a single source of truth for inheritance planning with collaborative, living plans that evolve with family needs, enabling transparent wealth transfer across generations.

Mission

Empower families to make informed inheritance decisions through transparency, collaboration, and professional guidance within a secure, compliant platform.

Market Analysis

Total Addressable Market (TAM)

- 11.8 million high-net-worth households in the US (>\$1M investable assets)💡
- Additional 20.3 million mass-affluent households (250K – 1M)💡
- **TAM: 32M households** 💡 *200/year average subscription* = **6.4B annual market**

Serviceable Addressable Market (SAM)

- Based on Cerulli Associates research, ~40% of HNW families actively use digital tools for financial planning💡
- Focus on digitally-engaged families planning wealth transfer
- **SAM: 12.8M households (40% of TAM) = \$2.56B annual market**

Serviceable Obtainable Market (SOM)

- 0.5% market penetration target based on comparable fintech SaaS adoption curve
- Conservative estimate accounting for 18-24 month sales cycle
- **SOM: 64,000 families = \$12.8M ARR by Year 3**

Competitive Landscape

| Feature | Forward | Vanilla | EstateMap | Wealth.com |
|--------------------------|---|----------------------------------|--------------------------|--------------------------|
| Family Collaboration | <input type="checkbox"/> Full | <input type="checkbox"/> | Partial | <input type="checkbox"/> |
| Marketing Independence | <input type="checkbox"/> Builder.io | Limited | Limited | <input type="checkbox"/> |
| Advisor Network | <input type="checkbox"/> Built-in | <input type="checkbox"/> Partner | <input type="checkbox"/> | <input type="checkbox"/> |
| HEI Integration | <input type="checkbox"/> Native | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Asset Diversity | <input type="checkbox"/> 13 types | Limited | Limited | Moderate |
| Individual Asset Control | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

User Research Insights

Based on market research and industry studies:

- **73%** of families report lack of transparency in inheritance planning
- **62%** want better collaboration tools with their advisors
- **Average estate settlement: 18 months** (opportunity to reduce by 50%)
- **70%** of wealth transfers fail by the third generation due to poor communication and trust breakdown

Sources:

1. Federal Reserve Survey of Consumer Finances, 2022
2. Spectrem Group Affluent Market Insights, 2023
3. Cerulli Associates: U.S. High-Net-Worth and Ultra-High-Net-Worth Markets 2023
4. Based on Bessemer Venture Partners SaaS adoption benchmarks for vertical fintech
5. Forward user research study (N=250 HNW families, Q3 2024)
6. Forward user research study (N=250 HNW families, Q3 2024)
7. EstateExec probate statistics, 2023
8. Williams Group wealth transfer research, 20-year longitudinal study

Strategic Insights

Target Market

- **Primary:** Mass-affluent/HNW families seeking tax-efficient wealth transfer and collaborative estate planning
- **Secondary:** Financial advisors specializing in estate planning and wealth transfer
- **Acquisition Strategy:** Direct marketing campaigns through [Builder.io](#) landing pages

Business Model

- **Revenue Streams:**
 - SaaS subscriptions (tiered family plans)
 - Advisor revenue sharing
 - Referral fees
 - Premium advisory services

- **Customer Journey:** Landing page ♦ registration ♦ FFC creation ♦ asset management ♦ advisor network expansion

Core Differentiation

- **Marketing Independence** through [Builder.io](#) CMS integration
- **Family-first transparency** vs individual portfolio focus
- **Individual asset control** - assets controlled by owners, not FFC owners
- **Comprehensive asset diversity** including alternative and digital assets
- **Enhanced security** with mandatory dual-channel verification
- **Living plans** that evolve with family circumstances
- **Database-first architecture** - all operations through stored procedures
- **Privacy-first documents** - automatic PII masking for document protection

Scale Target

- Millions of families with event-driven interaction patterns
- Setup, periodic check-ins, life events (births, deaths, marriages, divorces)
- Seasonal tax planning and annual reviews

Phase 1A Scope & Core Features

Phase 1A: True MVP Core (Days 1-60)

Goal: Launch with marketing foundation, complete FFC onboarding, HEI integration, and comprehensive asset management

Key Features for Launch:

1. Forward Marketing Foundation

- **Forward Landing Page** with [Builder.io](#) CMS integration
- **CMS Access** for marketing team independence
- **A/B Testing Capability** for conversion optimization

2. Complete FFC Onboarding Flow

- **User Registration** with email verification
- **Secure Login** with session management

- **FFC Creation** with unique naming
- **Member Invitation System** (email-based)
- **Mandatory Dual-Channel Verification** (email + SMS)
- **Owner Approval Required** for all new members
- **Mandatory Account Creation** for invited members
- **Role Assignment:**
 - FFC Owners (edit rights on FFC level)
 - Can add additional owners with edit rights
 - Other roles: Beneficiary, Non-beneficiary

3. Comprehensive Asset Management

- **All 13 Asset Categories** implemented from Day 1
- **Asset-Level Permissions:** Individual ownership rights per asset
- **Manual Asset Entry** for all asset types
- **HEI Integration** as part of Loan category (read-only from API)

4. Core Security & Infrastructure

- **Authentication & authorization**
- **Granular permissions** (FFC-level and asset-level)
- **Data encryption**
- **Audit logging for all changes**
- **Weekly audit reports** sent to FFC owners

Asset Categories (All 13 Types)

1. Personal Directives (POA, Healthcare Directive, Letter of Intent, HIPAA)
2. Trust
3. Will
4. Personal Property (Jewelry, Art, Pets, Furniture, etc.)
5. Operational Property (Vehicles, Boats, Equipment, Appliances)
6. Inventory
7. Real Estate
8. Life Insurance
9. Financial Accounts (Investments, Bank, Retirement, College)
10. Recurring Income (Royalties)
11. Digital Assets (IP, Digital Assets)
12. Ownership Interests (Business, Franchise)
13. **Loans (including HEI via API, Interfamily Loans)**

Key Permission Model

- **FFC Level:** Owners control FFC structure, invitations, approvals
- **Asset Level:** Each asset has independent ownership/permission structure
- **Critical Principle:** Asset owners control their assets, NOT FFC owners
- **Privacy First:** Assets only visible to personas explicitly granted permission

DEFERRED to Later Phases:

- L Chat System (removed entirely from MVP)
- L Referral Engine (moved to Phase 1B or later)
- L AI Suggestions
- L Document Intelligence

Epic Structure

Epic 1: Marketing Foundation & Landing Page

Epic Goal: Establish Forward's digital presence with a high-converting landing page that marketing can manage independently, enabling rapid iteration and A/B testing without developer involvement.

Business Value: Enable marketing-driven growth through autonomous content management, conversion optimization, and lead capture while development focuses on platform features.



Story 1.1: Landing Page Infrastructure Setup

As a developer

I want to configure [Builder.io](#) integration with our tech stack

So that marketers can create and publish content independently

Acceptance Criteria:

1. [Builder.io](#) account created and configured with Forward branding
2. [Builder.io](#) SDK integrated with React/TypeScript frontend
3. API keys securely stored in environment variables
4. Development and production spaces configured in [Builder.io](#)
5. Basic page templates created (home, about, contact)
6. Preview functionality working for marketers
7. Publish workflow established (draft  review  live)

8. Performance: Page loads in <2 seconds with [Builder.io](#) content

Technical Notes:

- Use [Builder.io](#) React SDK
- Implement error boundaries for [Builder.io](#) components
- Set up CDN caching for [Builder.io](#) content
- Create fallback content for API failures


Story 1.2: Marketing Landing Page Template

As a marketer

I want a customizable landing page template

So that I can create compelling content without developer help

Acceptance Criteria:

1. Hero section with customizable headline, subheadline, CTA buttons
2. Value proposition section with icon + text blocks
3. Feature showcase with image/video support
4. Social proof section (testimonials, logos)
5. Lead capture form integration
6. Footer with legal links and social media
7. Mobile-responsive design on all breakpoints
8. SEO metadata controls (title, description, OG tags)
9. Loading time  seconds on mobile networks

Design Requirements:

- Follow Forward brand guidelines
- Use existing design system components
- Ensure WCAG AA accessibility compliance
- Support for animations and micro-interactions

Story 1.3: Lead Capture & Form Integration

As a marketer

I want to capture and track visitor information

So that we can build our prospect database

Acceptance Criteria:

1. Customizable form builder in [Builder.io](#)
2. Form fields: Name, Email (required), Phone (optional), Interest (dropdown)
3. Email validation on client and server side
4. CAPTCHA integration to prevent spam
5. Form submissions stored in database
6. Email notification to sales team for hot leads
7. Thank you page/modal after submission
8. Integration with email marketing platform (SendGrid/Mailchimp)
9. GDPR compliance with consent checkboxes
10. Form conversion tracking in analytics

Technical Requirements:

- Secure form endpoint with rate limiting
- Honeypot field for bot detection
- Server-side validation
- Audit log for all submissions
- **All database operations through stored procedures**
- Database implementation details are documented in [architecture.md](#)
- See [architecture.md](#) section "Epic-Specific Stored Procedures > Epic 1: Marketing Foundation" for complete stored procedure specifications

Story 1.4: A/B Testing Framework

As a marketer

I want to run A/B tests on landing page elements

So that I can optimize conversion rates

Acceptance Criteria:

1. [Builder.io](#) A/B testing feature configured
2. Ability to create variant pages (A/B/n testing)
3. Test these elements: Headlines, CTAs, images, form fields
4. Traffic split configuration (50/50, 80/20, etc.)
5. Statistical significance calculator
6. Test duration controls (time or visitor based)
7. Automatic winner selection option
8. Test results dashboard in [Builder.io](#)
9. Integration with Google Analytics for deeper insights
10. QA mode to preview variants before launch

Success Metrics Setup:

- Page views and unique visitors
- Form submission rate
- CTA click-through rate
- Bounce rate by variant
- Time on page

Story 1.5: Analytics & Tracking Implementation

As a product manager

I want comprehensive analytics on landing page performance

So that we can make data-driven decisions

Acceptance Criteria:

1. Google Analytics 4 (GA4) properly configured
2. Google Tag Manager container setup
3. Conversion tracking for form submissions
4. Event tracking for key interactions:
 - CTA clicks
 - Video plays
 - Scroll depth
 - Form field interactions
5. UTM parameter tracking for campaigns
6. Heat mapping tool integration (Hotjar/Clarity)
7. Performance monitoring (Core Web Vitals)
8. Custom dashboard for marketing KPIs
9. Weekly automated reports to stakeholders

Privacy Requirements:

- Cookie consent banner implementation
- Privacy-compliant tracking (GDPR/CCPA)
- Option to opt-out of tracking
- Data retention policies configured

Story 1.6: SEO & Performance Optimization

As a marketer

I want our landing page to rank well and load fast

So that we can attract organic traffic and reduce bounce rates


Acceptance Criteria:

1. Server-side rendering (SSR) or static generation for SEO
2. Structured data markup (schema.org) for rich snippets
3. XML sitemap auto-generation
4. Robots.txt configuration
5. Canonical URL management
6. Image optimization with lazy loading
7. Core Web Vitals scoring:
 - LCP < 2.5s
 - FID < 100ms
 - CLS < 0.1
8. Mobile-friendly test passing
9. Meta tags manageable via [Builder.io](https://builder.io)
10. 301 redirect management system

Technical Optimizations:

- CDN configuration for static assets
- Brotli/gzip compression
- Browser caching headers
- Critical CSS inlining
- JavaScript bundle optimization

Epic 1 Success Criteria:

- ☐ Marketing team can publish content without developer help
- ☐ Landing page achieves >2% conversion rate
- ☐ Page load time <2 seconds on desktop,  seconds on mobile
- ☐ A/B testing running continuously
- ☐ Zero developer involvement needed for content updates
- ☐ Weekly performance reports automated
- ☐ 95%+ uptime for landing page

Epic 2: FFC Onboarding Flow

Epic Goal: Create a seamless, secure onboarding experience that guides users from registration through FFC creation, member invitation, and role assignment, establishing the foundation for family collaboration.

Business Value: Enable families to quickly form their Forward Family Circle with proper security, clear approval processes, and intuitive role management, driving platform adoption and engagement.

Story 2.1: User Registration & Account Creation

As a new user

I want to create my Forward account

So that I can start or join a family circle

Acceptance Criteria:

1. Registration form with fields:
 - First Name, Last Name (required)
 - Email (required, unique)
 - Password (required, 12+ chars, complexity rules)
 - Phone (optional)
 - Timezone auto-detection with manual override
2. Real-time validation with clear error messages
3. Email uniqueness check (instant feedback)
4. Password strength indicator
5. Terms of Service and Privacy Policy checkboxes (required)
6. Email verification sent within 30 seconds
7. Verification link expires after 24 hours
8. Welcome email sent after verification
9. Prevent registration with disposable email addresses
10. Account creation tracked in audit log

Technical Requirements:

- JWT token generation
- Secure password hashing (bcrypt/argon2)
- Rate limiting on registration endpoint
- Honeypot and CAPTCHA for bot prevention

Story 2.2: Secure Login & Session Management

As a registered user

I want to securely access my account

So that my family information remains protected

Acceptance Criteria:

1. Login form with email and password
2. "Remember me" option (30-day cookie)
3. Forgot password flow with secure reset
4. Account lockout after 5 failed attempts (30 min)
5. Session timeout after 24 hours of inactivity
6. Clear session status indicator
7. Secure logout clearing all tokens
8. Login from new device notification email
9. Optional 2FA setup (Phase 1B)
10. Support for password managers

Security Requirements:

- HTTPS only
- Secure session cookies
- CSRF protection
- Rate limiting per IP
- Audit log for all login attempts

Story 2.3: FFC Creation Workflow

As an authenticated user

I want to create my Forward Family Circle

So that I can begin organizing my family's assets

Acceptance Criteria:

1. Multi-step creation wizard:
 - Step 1: FFC Name (unique, 3-50 chars)
 - Step 2: Description (optional, 500 chars max)
 - Step 3: Creator becomes first Owner automatically
 - Step 4: Review and confirm
2. Real-time FFC name availability check

3. Suggested names if taken ("Smith Family" ❖ "Smith Family 2024")
4. Progress indicator showing current step
5. Save draft functionality between steps
6. Creation completes in ❤️ seconds
7. Success confirmation with next actions
8. FFC ID generated (user-friendly format: SMI-2024-ABC)
9. Audit log entry with timestamp
10. Redirect to member invitation screen

UI/UX Requirements:

- Mobile-responsive wizard
- Clear back/next navigation
- Validation before step progression
- Loading states for async operations

Story 2.4: Member Invitation System

As an FFC owner

I want to invite family members

So that they can participate in our family circle

Acceptance Criteria:

1. **Mandatory dual-channel setup:**
 - Both email and phone required (no optional fields)
 - Real-time phone format validation
 - International country code dropdown
 - Phone verification before invite sent
2. Invitation methods:
 - Email invitation (primary)
 - Bulk invite up to 20 emails/phones
3. Invitation includes:
 - FFC name and owner name
 - Personal message (optional, 200 chars)
 - Secure invitation token
 - Clear CTA to join
4. Invitation tracking:
 - Sent, Pending, Accepted, Expired status
 - Resend capability

- Cancel/revoke option
5. Invitations expire after 7 days
 6. Automatic reminder after 3 days
 7. Email delivery tracking
 8. Invitation history viewable by owner
 9. Template for invitation email (branded)
 10. Preview before sending
 11. Audit log for all invitations
 12. **Invitation template updates:**
 - Email explains phone verification requirement
 - Security warnings in both channels
 - Clear next steps for invitee

Technical Requirements:

- Unique invitation tokens
- Email service integration (SendGrid)
- SMS service integration (Twilio)
- Prevent invitation spam
- Rate limiting on invites

Story 2.5: Member Approval Process

As an FFC owner

I want to approve new members after verification

So that I control who joins our family circle

Acceptance Criteria:

1. Invitation acceptance flow:
 - Invited user clicks link ➡ Must verify phone ➡ Create account ➡ Request to join
 - Owner receives notification of pending approval
2. Owner approval dashboard:
 - List of pending approvals
 - Member details (name, email, phone, invitation date)
 - Verification status display
 - Approve/Deny buttons
 - Bulk approval option
3. Approval notifications:
 - Email to owner when someone requests

- Email to member when approved/denied
- In-app notifications (future)

4. Denied members:

- Polite denial message
- Can be re-invited later
- Reason for denial (internal note)

5. Approval timeout after 30 days

6. Co-owners can also approve members

7. Approval history log

8. **Post-verification approval:**

- Owner sees verification status before approving
- Approval triggers final SMS confirmation
- Invitee must confirm final SMS to gain access
- Two-step completion: approval + SMS confirmation

Business Rules:

- Only FFC owners can approve
- Owners cannot be removed by other owners
- At least one owner must remain

Story 2.6: Role Assignment & Management

As an FFC owner

I want to assign roles to members

So that everyone has appropriate access

Acceptance Criteria:

1. Available roles:

- Owner (full FFC edit rights)
- Beneficiary (read-only by default)
- Non-beneficiary (limited visibility)
- Advisor (professional access)

2. Role assignment during invitation

3. Role change after joining:

- Owner can change any member's role
- Clear UI showing current role
- Confirmation dialog for role changes

4. Owner-specific capabilities:

- Add another owner (co-owner)
- Cannot remove the last owner
- Transfer primary ownership (future)

5. Role change notifications:

- Email to affected member
- Audit log entry

6. Visual role indicators in member list

7. Role-based UI elements (hide/show features)

8. Bulk role assignment for efficiency

9. Role permissions clearly explained

10. Help documentation for each role

11. **Asset Permission Clarity:**

- Clear UI distinction between FFC roles and asset permissions
- Help text explaining: "FFC Owners manage the circle structure but do not automatically access member assets"
- Asset count shows as "X assets visible to you" not "X total assets"

12. **Permission Request Flow** (future enhancement):

- FFC Owners can request access to member assets
- Asset owners receive notification to grant/deny
- Maintains asset owner sovereignty

Updated Permission Matrix:

FFC-Level Permissions

| Action | Owner | Beneficiary | Non-beneficiary | Advisor |
|---------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| View FFC Structure | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Edit FFC Details | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Invite Members | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Approve Members | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Change Member Roles | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

Asset-Level Permissions (Controlled by Asset Owner)

| Action | Asset Owner | Granted Edit Rights | Granted View Rights | No Rights |
|--------------------|--------------------------|----------------------------|--------------------------|--------------------------|
| View Asset | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Edit Asset Details | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Change Ownership % | <input type="checkbox"/> | <input type="checkbox"/> * | <input type="checkbox"/> | <input type="checkbox"/> |
| Grant Permissions | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Delete Asset | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

*With asset owner approval

Story 2.7: Welcome & Onboarding Experience

As a new FFC member

I want to understand how to use Forward

So that I can effectively participate

Acceptance Criteria:

- 1. First-time user experience:
 - Welcome modal explaining Forward
 - 3-step visual tour of key features
 - Skip option available
- 2. Role-specific onboarding:
 - Owners: How to add assets, invite members
 - Beneficiaries: How to view assets, permissions
 - Advisors: How to access client information
- 3. Progress indicators for profile completion
- 4. Helpful empty states:
 - "Add your first asset" CTA
 - "Invite family members" prompt
- 5. Resource center links:
 - Video tutorials
 - Help documentation
 - FAQs

6. Onboarding completion tracking
7. Re-triggerable onboarding
8. Mobile-optimized onboarding




Story 2.8: Dual-Channel Phone Verification (MANDATORY)

As an FFC owner

I want mandatory phone verification for all invitees

So that only legitimate family members can join our circle

Final Security Flow: Dual-Channel Verification

1. FFC Owner invites  MUST provide email + phone (both required)
2. System sends:
 - Email: Invitation link + FFC details
 - SMS: 6-digit verification code + owner name
3. Invitee must complete ALL steps:
 - Click email link
 - Enter SMS verification code (10 min expiration)
 - Create account or login
 - Request to join FFC
4. Owner approves  System sends final SMS confirmation
5. Invitee confirms final SMS  Access granted

Acceptance Criteria:


1. Invitation Requirements (Owner Side):

- Email field (required, validated)
- Phone field (required, international format support)
- Cannot send invite without both
- Preview of dual-channel messages before sending

2. Dual-Channel Delivery:

- Email and SMS sent simultaneously
- SMS format: "Join [FFC Name] - Code: 123456 - From: [Owner Name] - Never share this code"
- Email includes security warning about phone verification
- Both messages expire simultaneously

3. Verification Process:

- Link from email  Landing page requesting SMS code
- Must enter correct 6-digit code to proceed

- 3 attempts maximum, then 30-minute lockout
- Clear error messages for incorrect codes
- Timer showing remaining time for code

4. **Phone Number Sharing (Edge Case Support):**

- Same phone number allowed for multiple FFC members
- SMS includes member name when sent
- Verification page shows "Joining as: [Name]"
- Audit log tracks which member used shared phone

5. **Security Features:**

- Code single-use only
- 10-minute expiration (strict)
- Owner notified of failed verification attempts
- All verification attempts logged
- Rate limiting: 5 codes per phone per hour

6. **Cost Management:**

- SMS costs absorbed by platform
- International SMS supported
- WhatsApp backup for countries with poor SMS
- Voice call option for accessibility

7. **Error Handling:**

- Failed SMS delivery ⚡ Voice call backup
- Invalid phone numbers ⚡ Clear error to owner
- Network timeouts ⚡ Retry mechanism
- Graceful degradation for service outages

Technical Implementation Notes:

- **SMS Provider:** Twilio for global coverage
- **Phone Validation:** libphonenumber for format checking
- **Backup Delivery:** Voice calls via Twilio Voice
- **International Support:** WhatsApp Business API for regions with poor SMS
- **Security:** All phone numbers encrypted at rest
- **Audit:** Complete verification trail for compliance

Epic 2 Success Criteria:

- ☐ 90%+ registration completion rate
- ☐ Email verification rate >85%
- ☐ FFC creation completion >80% of started

- ☐ Invitation acceptance rate >40%
- ☐ Member approval time <24 hours average
- ☐ Zero security breaches
- ☐ Verification completion rate >90% (both email + SMS)
- ☐ Onboarding NPS score >4.0/5
- ☐ False positive rate <1% (legitimate members blocked)
- ☐ Security incident rate: 0 successful phishing attempts

Technical Architecture

Core Architecture Principles

Mandatory Database Access Layer

All database operations MUST use stored procedures and functions:

- No direct table access from application code
- All CRUD operations through stored procedures
- Complex queries through database functions
- Business logic enforcement at database level
- Type-safe operations with pgtyped
- Comprehensive audit logging at procedure level

Benefits:

1. **Security:** SQL injection prevention, access control
2. **Performance:** Optimized execution plans, reduced network traffic
3. **Consistency:** Centralized business rules and validation
4. **Maintainability:** Database API versioning, easier refactoring
5. **Compliance:** Automatic audit trails and data governance

Frontend Technology Stack

Core Framework and Tools

- **React 18+ with TypeScript**
 - Functional components with hooks
 - Strict TypeScript configuration for type safety
 - React Router v6 for client-side routing

- React Query for server state management
- **Styling and UI Framework**
 - Tailwind CSS for utility-first styling
 - shadcn/ui component library for consistent design
 - CSS modules for component-specific styles
 - Design system implementation from existing mockups
- **Build and Development Tools**
 - Vite for fast development and building
 - ESLint and Prettier for code quality
 - Husky for pre-commit hooks
 - Testing with Jest and React Testing Library

Application Architecture

- **Mobile-First Responsive Design**
 - Progressive enhancement approach
 - Breakpoint-based responsive design
 - Touch-friendly interface optimization
 - Cross-browser compatibility (modern browsers)
- **Single Page Application (SPA)**
 - Client-side routing and navigation
 - Code splitting for optimal bundle sizes
 - Lazy loading for performance optimization
 - Progressive Web App (PWA) capabilities
- **Component Architecture**
 - Card-based modular component system
 - Reusable component library
 - Atomic design principles
 - Storybook for component documentation
- **State Management**
 - Context API for global application state
 - React Query for server state caching
 - Local storage for user preferences
 - Session management for authentication

Integration Features

- **Builder.io CMS Integration**
 - Headless CMS for marketing content

- Dynamic content delivery
- A/B testing capabilities
- SEO optimization
- **File Management**
 - Document upload and preview
 - Image optimization and compression
 - Secure file sharing and access control
 - Version control for documents

Backend Technology Stack

Core Server Framework

- **Node.js with Express and TypeScript**
 - RESTful API architecture
 - GraphQL for complex data queries (future consideration)
 - Middleware for authentication, logging, and error handling
 - Rate limiting and security headers
- **Database and ORM**
 - PostgreSQL for primary data storage
 - Slonik for safe PostgreSQL client operations
 - pgtyped for compile-time SQL type safety
 - Database migrations and version control
- **Business Logic Architecture**
 - Stored procedures and functions for complex operations
 - Transaction management for data consistency
 - Event-driven architecture for system integration
 - Background job processing with Bull/Agenda

Integration Infrastructure

- **Financial Data Integration**
 - RESTful API clients for financial service providers
 - Real-time webhook handling for account synchronization
 - Data transformation and mapping services
 - Conflict resolution and error handling
- **Email Service Integration**
 - SendGrid for transactional emails
 - Template management for referral campaigns

- Bounce and complaint handling
- Email analytics and tracking
- **SMS Service Integration**
 - Twilio for SMS delivery and phone verification
 - WhatsApp Business API for international support
 - Voice call backup for verification
 - Phone number validation and formatting

Security and Authentication

- **Authentication and Authorization**
 - JWT-based authentication
 - Role-based access control (RBAC)
 - Multi-factor authentication (2FA)
 - Session management and security
- **Data Protection**
 - Encryption at rest and in transit
 - PII masking and data anonymization
 - GDPR and CCPA compliance
 - Audit logging for all data access
- **API Security**
 - Rate limiting and DDoS protection
 - Input validation and sanitization
 - CORS and security headers
 - API versioning and deprecation management

AWS Cloud Migration Roadmap

Phase 1: Basic Cloud Infrastructure

- **Content Delivery**
 - CloudFront for global CDN
 - S3 for static asset storage
 - Route 53 for DNS management
 - SSL/TLS certificate management
- **Application Hosting**
 - Amplify for frontend CI/CD and hosting
 - Alternative: S3 + CloudFront for static site hosting
 - API Gateway for secure, scalable API endpoints

- Lambda functions for serverless operations

Phase 2: Containerized Services

- **Container Orchestration**
 - EKS (Elastic Kubernetes Service) for container management
 - Fargate for serverless container execution
 - Docker containerization for all services
 - Auto-scaling based on demand
- **Database Services**
 - RDS PostgreSQL for development and testing
 - Aurora PostgreSQL for production scalability
 - ElastiCache for Redis caching layer
 - Database backup and disaster recovery

Security and Compliance

Compliance Frameworks

- **SOC 1 & SOC 2 Compliance**
 - Regular third-party audits
 - Control implementation and monitoring
 - Risk assessment and mitigation
 - Continuous compliance monitoring
- **Industry Standards**
 - PCI DSS for payment processing
 - GDPR for data protection
 - CCPA for California privacy rights
 - HIPAA considerations for health directives
- **Penetration Testing**
 - Regular security assessments
 - Vulnerability scanning and remediation
 - Third-party security audits
 - Bug bounty program considerations

Security Infrastructure

- **Vanta Integration**
 - Continuous compliance monitoring
 - Automated security assessments

- Policy management and enforcement
- Audit trail maintenance
- **Access Controls**
 - Two-factor authentication enforcement
 - Single sign-on (SSO) integration
 - Session management and timeout
 - IP whitelisting and geo-blocking
- **Data Protection**
 - Context-aware access controls
 - Dynamic PII masking
 - Data loss prevention (DLP)
 - Encryption key management
- **Network Security**
 - VPC segmentation and isolation
 - Network access control lists (NACLs)
 - Security groups and firewall rules
 - VPN access for administrative functions

Performance Requirements

API Performance Targets

- **Response Times**
 - p50: < 100ms
 - p95: < 200ms
 - p99: < 500ms
- **Throughput**
 - 10,000 concurrent users
 - 100 financial sync operations/second

Frontend Performance

- **Page Load Times**
 - First Contentful Paint (FCP): < 1.5s
 - Time to Interactive (TTI): < 3.5s
 - Cumulative Layout Shift (CLS): < 0.1
- **Mobile Performance**
 - Touch response: < 100ms
 - Smooth scrolling: 60fps

- Offline capability for critical features

Database Performance

- **Query Performance**
 - Simple queries: < 10ms
 - Complex queries: < 50ms
 - Bulk operations: < 500ms
- **Connection Management**
 - Connection pool: 100-500 connections
 - Query timeout: 30s
 - Transaction timeout: 60s

Database Schema

Complete database schema designs are documented in [architecture.md](#).

Key Components:

- **Multi-Tenant Core Schemas:** Tenants, Personas, FFCs, Contact/Communication
- **Enhanced Asset Management:** 13 asset categories with ownership and permissions
- **Invitation & Verification:** Dual-channel verification system
- **Audit & Compliance:** Comprehensive audit trails and security measures
- **Performance Indexes:** Optimized for multi-tenant operations

Reference Sections in [architecture.md](#):

- **Multi-Tenant Core Schemas:** Complete table definitions for tenants, personas, FFCs
- **Contact and Communication Schema:** Flexible address, phone, and email management
- **Enhanced Asset Management Schema:** All 13 asset categories with ownership models
- **Invitation and Verification Schema:** Dual-channel security verification
- **Enhanced Audit and Compliance Schema:** Comprehensive audit logging
- **Performance Indexes:** All database indexes for optimal performance

See [architecture.md](#) for complete table definitions, triggers, constraints, and indexes.

Success Metrics

Launch Metrics (Day 1-30)

- **Registration Completion Rate:** >85%
 - Measured from landing page to verified account
 - A/B test onboarding variations
- **FFC Creation Rate:** >70% of registered users
 - Time from registration to first FFC
 - Track abandonment points
- **First Asset Added:** >60% of FFCs
 - Manual entry vs. import usage
 - Category distribution analysis
- **Member Invites Sent:** Average 3 per FFC
 - Dual-channel verification completion rate
 - Acceptance rate tracking
 - Time to acceptance
- **Invite Acceptance Rate:** >40%
 - By relationship type
 - Re-invitation success
- **Phone Verification Success:** >90%
 - SMS delivery rate
 - Code entry success rate
 - International vs domestic performance

Growth Metrics (Day 31-90)

- **Weekly Active Family Circles (WAFCs):** 60%
 - Definition: 2+ members, 3+ actions/week
 - Growth target: 10% WoW
- **Assets per FFC:** Average 5+
 - Distribution by category
 - Value concentration analysis
- **Landing Page Performance:**
 - Conversion rate >2%
 - Page load time <2 seconds
 - A/B test winners implemented
- **Marketing Independence:** 100% of content updates by marketing team

- **Asset Permission Changes:** Track frequency and patterns
- **Security Metrics:** Zero successful phishing attempts

North Star Metric

Weekly Active Family Circles (WAFCs)

- **Definition:** FFCs with 2+ members and 3+ platform actions per week
- **Target:** 50% of all FFCs are WAFCs by Day 90
- **Growth:** 10% week-over-week increase in WAFCs
- **Why it matters:** Indicates true platform value and stickiness

Business KPIs

Platform Foundation Success

- **Target:** 95% successful family onboarding completion
- **Measurement:** End-to-end flow completion rate
- **Sub-metrics:**
 - Setup abandonment: <5%
 - Time to complete: <15 minutes
 - User satisfaction: >4.0/5

Security Performance

- **Verification Success Rate:** >90%
- **Phone Verification Completion:** >85%
- **False Positive Rate:** <1%
- **Security Incident Rate:** 0

Marketing Performance

- **Landing Page Load Time:** <2 seconds
- **Form Conversion Rate:** >2%
- **Content Update Frequency:** Daily capability
- **A/B Test Velocity:** 2+ tests per month

Risk Mitigation

Risk Scoring Matrix

| Risk | Probability | Impact | Score | Priority | Mitigation Strategy |
|---|--------------|--------|-------|----------|---|
| Phone Verification Delivery | Medium (40%) | High | 16 | P0 | Multiple SMS providers, voice backup, WhatsApp integration |
| Builder.io Performance Impact | Medium (30%) | Medium | 9 | P1 | CDN optimization, fallback content, performance monitoring |
| Asset Permission Complexity | High (60%) | Medium | 18 | P0 | Clear UI, extensive testing, user education, progressive disclosure |
| Invitation Fraud/Abuse | Low (20%) | High | 12 | P1 | Rate limiting, phone verification, audit trails, manual review |
| International SMS Costs | High (70%) | Low | 7 | P2 | WhatsApp backup, voice alternatives, cost monitoring |

Security Risks

Phone Verification System

Risk: SMS delivery failures or phone number spoofing

Mitigation:

- Multi-provider SMS strategy (Twilio primary, backup providers)
- Voice call verification as backup
- WhatsApp Business API for international
- Rate limiting and abuse detection
- Complete audit trail for all verification attempts

Asset Permission System

Risk: Complex permissions leading to user errors or security gaps

Mitigation:

- Progressive disclosure of advanced features
- Clear visual indicators for permission levels
- Default to secure permissions
- Permission preview before saving
- Extensive user testing and iterative improvement

Invitation Security

Risk: Phishing or unauthorized access attempts

Mitigation:

- Dual-channel verification required
- Unique tokens with expiration
- Owner approval after verification
- Comprehensive audit logging
- Clear security messaging to users

Technical Risks

Builder.io Integration

Risk: Performance impact or service dependency

Mitigation:

- CDN caching for all [Builder.io](#) content
- Fallback static content for outages
- Performance monitoring and alerting
- Regular performance testing
- Alternative CMS evaluation

Database Performance

Risk: Complex permission queries causing slowdowns

Mitigation:

- Optimized indexes for permission checks
- Query performance monitoring

- Caching layer for frequently accessed data
- Database scaling plan
- Regular performance reviews

Business Risks

User Adoption of Security Features

Risk: Complex verification process reducing adoption

Mitigation:

- Clear explanation of security benefits
- Progressive onboarding
- Support team training
- User feedback collection
- Iterative UX improvements

Marketing Team [Builder.io](#) Learning Curve

Risk: Delayed marketing capability due to tool complexity

Mitigation:

- Comprehensive [Builder.io](#) training
- Template library for common use cases
- Dedicated support channel
- Documentation and video tutorials
- Gradual feature rollout

Early Warning System

Monitoring Triggers:

- SMS delivery rate <85% ⚡ Switch providers
- Landing page load time >3s ⚡ Performance investigation
- Permission errors >5/day ⚡ UI review
- Failed verification attempts >10% ⚡ Security review
- Asset creation abandonment >30% ⚡ UX analysis

Epic Structure & Implementation

Epic 1: Marketing Foundation & Landing Page

Goal: Establish marketing independence through [Builder.io](#) CMS integration and high-converting landing pages.

Duration: 2 weeks (Sprints 1-2)

Story 1.1: [Builder.io](#) CMS Integration

As a marketing team member

I want to manage landing page content independently

So that we can iterate quickly without developer dependencies

Acceptance Criteria:

- [Builder.io](#) workspace configured with Forward branding
- Landing page templates created for different campaigns
- Visual editor accessible to marketing team
- Content changes reflect in production within 5 minutes
- A/B testing capabilities enabled

Story 1.2: Landing Page Performance & SEO

As a visitor

I want fast-loading, discoverable landing pages

So that I can quickly learn about Forward's services

Acceptance Criteria:

- Page load time <1.5 seconds (First Contentful Paint)
- Google PageSpeed score >90
- SEO meta tags and structured data implemented
- Mobile responsiveness across all devices
- Analytics tracking configured (Google Analytics, Hotjar)

Story 1.3: Lead Capture System

As a potential customer

I want to easily express interest and get follow-up

So that I can learn more about Forward's services

Acceptance Criteria:

- Contact form with progressive profiling
- Email validation and spam protection
- CRM integration (HubSpot/Salesforce)
- Automated follow-up email sequence
- Lead scoring and qualification workflow

Database Implementation:

All database operations must use stored procedures:

```

-- Lead capture stored procedure
CREATE OR REPLACE FUNCTION capture_marketing_lead(
    p_name VARCHAR,
    p_email VARCHAR,
    p_phone VARCHAR,
    p_interest VARCHAR,
    p_utm_source VARCHAR,
    p_utm_campaign VARCHAR,
    p_ip_address INET
) RETURNS TABLE(
    lead_id UUID,
    status VARCHAR,
    message VARCHAR
) AS $$
DECLARE
    v_lead_id UUID;
    v_existing_lead UUID;
BEGIN
    -- Check for existing lead
    SELECT id INTO v_existing_lead
    FROM marketing_leads
    WHERE email = p_email;

    IF v_existing_lead IS NOT NULL THEN
        -- Update existing lead
        UPDATE marketing_leads
        SET
            updated_at = NOW(),
            last_interest = p_interest,
            utm_source = p_utm_source,
            utm_campaign = p_utm_campaign,
            contact_attempts = contact_attempts + 1
        WHERE id = v_existing_lead;

        RETURN QUERY SELECT v_existing_lead, 'updated'::VARCHAR, 'Lead updated successfully'::V
    ELSE
        -- Create new lead
        INSERT INTO marketing_leads (
            name, email, phone, interest, utm_source, utm_campaign, ip_address
        ) VALUES (
            p_name, p_email, p_phone, p_interest, p_utm_source, p_utm_campaign, p_ip_address
        ) RETURNING id INTO v_lead_id;

```

```
        RETURN QUERY SELECT v_lead_id, 'created'::VARCHAR, 'Lead captured successfully'::VARCHAR;  
    END IF;  
END;  
$$ LANGUAGE plpgsql;
```

Epic 2: FFC Onboarding Flow with Enhanced Security

Goal: Enable secure family circle creation with mandatory dual-channel verification.

Duration: 3 weeks (Sprints 3-5)

Story 2.1: User Registration & Email Verification

As a new user

I want to create an account securely

So that I can start organizing my family's wealth

Acceptance Criteria:

- Strong password requirements (12+ chars, complexity)
- Email verification required before account activation
- Account lockout after 5 failed attempts
- Password reset functionality
- Terms of service and privacy policy acceptance

Database Implementation:

Database implementation details are documented in [architecture.md](#)

See [architecture.md](#) section "Epic-Specific Stored Procedures > Epic 2: FFC Onboarding" for complete stored procedure specifications including `register_user` and `invite_ffc_member` functions.

Story 2.2: FFC Creation Wizard

As a verified user

I want to create my family forward circle easily

So that I can start collaborating with family members

Acceptance Criteria:

- Multi-step wizard with progress indicator
- FFC name uniqueness validation
- Family description and goals capture

- Initial role assignment (Owner)
- Success confirmation with next steps

Story 2.3: Enhanced Member Invitation System

As an FFC owner

I want to invite family members with dual-channel verification

So that we ensure maximum security for sensitive family data

Acceptance Criteria:

- Email invitation with secure token
- Mandatory phone number verification (SMS)
- Invite expiration after 7 days
- Role assignment during invitation
- Owner approval required for all invitations

Database implementation details are documented in [architecture.md](#)

See [architecture.md](#) section "Epic-Specific Stored Procedures > Epic 2: FFC Onboarding" for complete `invite_ffc_member` stored procedure specification.

Story 2.4: Phone Verification System

As an invited family member

I want to verify my phone number

So that the FFC owner can be confident in my identity

Acceptance Criteria:

- SMS code sent via Twilio
- 6-digit numeric code format
- Code expires after 10 minutes
- Maximum 3 verification attempts
- Fallback to voice call option

Story 2.5: Owner Approval Workflow

As an FFC owner

I want to approve member invitations after verification

So that I maintain control over who joins my family circle

Acceptance Criteria:

- Notification to owner when verification complete
- Approval/rejection interface with reason capture
- Automatic approval option (configurable)
- Member onboarding email after approval
- Activity log for all approval decisions

Epic 3: Comprehensive Asset Management System

Goal: Enable secure tracking and management of all 13 asset categories with document support and PII protection.

Duration: 4 weeks (Sprints 6-9)

Database Implementation: All stored procedures for asset management, document handling, and PII processing are documented in [architecture.md](#) section "Epic-Specific Stored Procedures > Epic 3: Asset Management & PII Protection"

Story 3.1: Asset Category Infrastructure

As a system architect

I want robust asset category support

So that users can track all types of family wealth

Acceptance Criteria:

- All 13 asset categories implemented
- Category-specific database schemas
- Dynamic form generation per category
- Validation rules per asset type
- Category icons and descriptions

Story 3.2: Asset-Persona Ownership Model

As an asset owner

I want direct ownership tracking

So that my assets are clearly attributed to me

Acceptance Criteria:

- Direct persona-to-asset ownership links

- Percentage ownership support (0.01% precision)
- Ownership type classification (direct, trust, beneficiary)
- Ownership transfer functionality
- Ownership history tracking

Database Implementation:

```

-- Asset creation stored procedure
CREATE OR REPLACE FUNCTION create_asset_with_ownership(
    p_owner_persona_id UUID,
    p_category_id INTEGER,
    p_asset_data JSONB,
    p_ownership_percentage DECIMAL(5,2) DEFAULT 100.00,
    p_ownership_type ownership_type_enum DEFAULT 'direct'
) RETURNS TABLE(
    asset_id UUID,
    ownership_id UUID,
    status VARCHAR
) AS $$
DECLARE
    v_asset_id UUID;
    v_ownership_id UUID;
    v_category_table TEXT;
BEGIN
    -- Validate ownership percentage
    IF p_ownership_percentage <= 0 OR p_ownership_percentage > 100 THEN
        RETURN QUERY SELECT NULL::UUID, NULL::UUID, 'invalid_percentage'::VARCHAR;
        RETURN;
    END IF;

    -- Create main asset record
    INSERT INTO assets (
        category_id, created_by_persona_id, status
    ) VALUES (
        p_category_id, p_owner_persona_id, 'active'
    ) RETURNING id INTO v_asset_id;

    -- Insert category-specific data
    SELECT table_name INTO v_category_table
    FROM asset_categories
    WHERE id = p_category_id;

    EXECUTE format('
        INSERT INTO %I (asset_id, data)
        VALUES ($1, $2)', v_category_table)
    USING v_asset_id, p_asset_data;

    -- Create ownership record
    INSERT INTO asset_persona_ownership (
        asset_id, persona_id, ownership_percentage, ownership_type

```

```
) VALUES (  
    v_asset_id, p_owner_persona_id, p_ownership_percentage, p_ownership_type  
) RETURNING id INTO v_ownership_id;  
  
RETURN QUERY SELECT v_asset_id, v_ownership_id, 'success'::VARCHAR;  
END;  
$$ LANGUAGE plpgsql;
```

Story 3.3: Document & Photo Management with PII Protection

As an asset owner

I want to upload supporting documents and photos

So that I can maintain complete asset records while protecting sensitive information

Acceptance Criteria:

- Multiple file uploads per asset
- PDF, JPG, PNG, DOC, XLS format support
- Files up to 10MB each, 100MB total per asset
- Document categorization (deed, statement, photo, etc.)
- Original document encryption in S3
- Automatic PII detection and masking
- Separate storage for PII-masked versions

PII Masking System Architecture:


```

-- Document upload with PII masking stored procedure
CREATE OR REPLACE FUNCTION upload_asset_document(
    p_asset_id UUID,
    p_uploader_persona_id UUID,
    p_filename VARCHAR,
    p_file_size BIGINT,
    p_file_type VARCHAR,
    p_document_category VARCHAR,
    p_s3_key_original VARCHAR,
    p_extracted_text TEXT DEFAULT NULL
) RETURNS TABLE(
    document_id UUID,
    processing_job_id VARCHAR,
    status VARCHAR
) AS $$
DECLARE
    v_document_id UUID;
    v_job_id VARCHAR;
    v_contains_pii BOOLEAN := FALSE;
BEGIN
    -- Validate file size (10MB limit)
    IF p_file_size > 10485760 THEN
        RETURN QUERY SELECT NULL::UUID, NULL::VARCHAR, 'file_too_large'::VARCHAR;
        RETURN;
    END IF;

    -- Create document record
    INSERT INTO asset_documents (
        asset_id, uploader_persona_id, filename, file_size,
        file_type, document_category, s3_key_original,
        extracted_text, processing_status
    ) VALUES (
        p_asset_id, p_uploader_persona_id, p_filename, p_file_size,
        p_file_type, p_document_category, p_s3_key_original,
        p_extracted_text, 'processing'
    ) RETURNING id INTO v_document_id;

    -- Queue PII detection job
    v_job_id := 'pii_' || v_document_id::TEXT || '_' || extract(epoch from now())::TEXT;

    INSERT INTO pii_processing_jobs (
        job_id, document_id, status, queued_at
    ) VALUES (

```

```

        v_job_id, v_document_id, 'queued', NOW()
    );

    RETURN QUERY SELECT v_document_id, v_job_id, 'queued_for_processing'::VARCHAR;
END;
$$ LANGUAGE plpgsql;

-- PII masking completion stored procedure
CREATE OR REPLACE FUNCTION complete_pii_processing(
    p_job_id VARCHAR,
    p_pii_entities JSONB,
    p_masked_text TEXT,
    p_s3_key_masked VARCHAR,
    p_confidence_score DECIMAL(3,2)
) RETURNS TABLE(
    document_id UUID,
    pii_detected BOOLEAN,
    status VARCHAR
) AS $$
DECLARE
    v_document_id UUID;
    v_pii_detected BOOLEAN;
BEGIN
    -- Get document ID from job
    SELECT document_id INTO v_document_id
    FROM pii_processing_jobs
    WHERE job_id = p_job_id;

    IF v_document_id IS NULL THEN
        RETURN QUERY SELECT NULL::UUID, FALSE, 'job_not_found'::VARCHAR;
        RETURN;
    END IF;

    -- Determine if PII was detected
    v_pii_detected := (jsonb_array_length(p_pii_entities) > 0);

    -- Update document with PII processing results
    UPDATE asset_documents SET
        pii_entities = p_pii_entities,
        masked_text = p_masked_text,
        s3_key_masked = p_s3_key_masked,
        pii_confidence_score = p_confidence_score,
        contains_pii = v_pii_detected,

```

```

        processing_status = 'completed',
        processed_at = NOW()
WHERE id = v_document_id;

-- Update job status
UPDATE pii_processing_jobs SET
    status = 'completed',
    completed_at = NOW()
WHERE job_id = p_job_id;

-- Log PII detection event
INSERT INTO audit_log (
    user_id, action, details, created_at
) VALUES (
    (SELECT p.user_id FROM personas p
     JOIN asset_documents ad ON p.id = ad.uploader_persona_id
     WHERE ad.id = v_document_id),
    'pii_processing_completed',
    jsonb_build_object(
        'document_id', v_document_id,
        'pii_detected', v_pii_detected,
        'entities_count', jsonb_array_length(p_pii_entities),
        'confidence_score', p_confidence_score
    ),
    NOW()
);

RETURN QUERY SELECT v_document_id, v_pii_detected, 'success'::VARCHAR;
END;
$$ LANGUAGE plpgsql;

```

PII Detection Pipeline:

1. **Upload:** Original document encrypted and stored in S3
2. **Text Extraction:** OCR/text parsing for searchable content
3. **PII Detection:** AWS Comprehend identifies SSNs, phone numbers, addresses, names
4. **Masking:** Sensitive data replaced with tokens (XXX-XX-1234)
5. **Dual Storage:** Original (encrypted) + masked version (encrypted)
6. **Access Control:** Role-based access to original vs masked versions

Supported PII Types:

- Social Security Numbers

- Phone Numbers
- Email Addresses
- Physical Addresses
- Financial Account Numbers
- Names and Personal Identifiers
- Date of Birth
- Driver's License Numbers

Story 3.4: Individual Asset Permissions

As an asset owner

I want granular control over who can see my assets

So that I maintain privacy within my family circle

Acceptance Criteria:

- Read/Edit/Admin permission levels
- Per-asset permission assignment
- Inheritance of FFC-level permissions (optional)
- Permission override capabilities
- Bulk permission management
- Permission audit trail

Story 3.5: Asset Dashboard & Visualization

As an FFC member

I want clear visualization of family assets

So that I understand our collective wealth picture

Acceptance Criteria:

- Category-based asset grouping
- Total value calculations by category
- Visual charts and graphs
- Filter by ownership, category, value
- Export capabilities (PDF, Excel)
- Mobile-responsive design

Story 3.6: HEI Integration (Read-Only)

As a family with HEI loans

I want to see loan information in our asset dashboard

So that we have a complete financial picture

Acceptance Criteria:

- HEI API integration for loan data
- Read-only display in loan category
- Automatic data synchronization
- Integration with permission system
- Loan performance metrics display

Epic 4: Advanced Features & Integrations

Goal: Enhance platform capabilities with advanced search, reporting, audit trails, and third-party integrations.

Duration: 3 weeks (Sprints 10-12)

Database Implementation: All stored procedures for search, reporting, audit trails, integrations, and Quillt API functions are documented in [architecture.md](#) section "Epic-Specific Stored Procedures > Epic 4: Reporting, Analytics & Integrations" and "Integration Architecture > Quillt Integration Stored Procedures"

Story 4.1: Advanced Search & Filtering System

As an FFC member

I want powerful search capabilities across all assets and documents

So that I can quickly find specific information within our family wealth

Acceptance Criteria:

- Global search across all asset categories
- Advanced filters (category, value range, owner, date)
- Search within document content (OCR text)
- Saved search queries
- Search result sorting and pagination
- Search analytics and suggestions

Database Implementation:

```

-- Advanced search stored procedure
CREATE OR REPLACE FUNCTION search_family_assets(
    p_ffc_id UUID,
    p_search_query TEXT,
    p_category_filters INTEGER[],
    p_owner_filters UUID[],
    p_value_min DECIMAL DEFAULT NULL,
    p_value_max DECIMAL DEFAULT NULL,
    p_date_from DATE DEFAULT NULL,
    p_date_to DATE DEFAULT NULL,
    p_limit INTEGER DEFAULT 50,
    p_offset INTEGER DEFAULT 0
) RETURNS TABLE(
    asset_id UUID,
    asset_name VARCHAR,
    category_name VARCHAR,
    owner_name VARCHAR,
    estimated_value DECIMAL,
    match_score DECIMAL,
    snippet TEXT
) AS $$
DECLARE
    v_search_vector tsvector;
    v_query tsquery;
BEGIN
    -- Convert search query to full-text search
    v_query := plainto_tsquery('english', p_search_query);

    RETURN QUERY
    WITH asset_search AS (
        SELECT
            a.id as asset_id,
            a.name as asset_name,
            ac.name as category_name,
            p.first_name || ' ' || p.last_name as owner_name,
            a.estimated_value,
            ts_rank(a.search_vector, v_query) as match_score,
            ts_headline('english', a.description, v_query) as snippet
        FROM assets a
        JOIN asset_categories ac ON a.category_id = ac.id
        JOIN asset_persona_ownership apo ON a.id = apo.asset_id
        JOIN personas p ON apo.persona_id = p.id
        JOIN ffc_personas fp ON p.id = fp.persona_id
    )

```

```

WHERE fp.ffc_id = p_ffc_id
AND (v_query IS NULL OR a.search_vector @@ v_query)
AND (p_category_filters IS NULL OR a.category_id = ANY(p_category_filters))
AND (p_owner_filters IS NULL OR p.id = ANY(p_owner_filters))
AND (p_value_min IS NULL OR a.estimated_value >= p_value_min)
AND (p_value_max IS NULL OR a.estimated_value <= p_value_max)
AND (p_date_from IS NULL OR a.created_at >= p_date_from)
AND (p_date_to IS NULL OR a.created_at <= p_date_to)
ORDER BY match_score DESC, a.estimated_value DESC
LIMIT p_limit OFFSET p_offset
)
SELECT * FROM asset_search;
END;
$$ LANGUAGE plpgsql;

```

Story 4.2: Comprehensive Reporting & Analytics

As an FFC owner

I want detailed reports on our family wealth

So that I can make informed financial decisions and track changes over time

Acceptance Criteria:

- Wealth distribution reports by category and person
- Asset growth tracking over time
- PDF report generation with charts
- Scheduled report delivery (email)
- Custom report builder
- Export to Excel/CSV formats

Database Implementation:

```

-- Wealth summary report stored procedure
CREATE OR REPLACE FUNCTION generate_wealth_report(
    p_ffc_id UUID,
    p_report_date DATE DEFAULT CURRENT_DATE,
    p_include_projections BOOLEAN DEFAULT FALSE
) RETURNS TABLE(
    category_name VARCHAR,
    asset_count INTEGER,
    total_value DECIMAL,
    avg_value DECIMAL,
    top_owner VARCHAR,
    growth_12m_percent DECIMAL
) AS $$
BEGIN
    RETURN QUERY
    WITH wealth_summary AS (
        SELECT
            ac.name as category_name,
            COUNT(a.id)::INTEGER as asset_count,
            COALESCE(SUM(a.estimated_value), 0) as total_value,
            COALESCE(AVG(a.estimated_value), 0) as avg_value,
            (
                SELECT p.first_name || ' ' || p.last_name
                FROM personas p
                JOIN asset_persona_ownership apo ON p.id = apo.persona_id
                JOIN assets a2 ON apo.asset_id = a2.id
                WHERE a2.category_id = ac.id
                AND EXISTS (
                    SELECT 1 FROM ffc_personas fp
                    WHERE fp.persona_id = p.id AND fp.ffc_id = p_ffc_id
                )
            ) as top_owner,
            COALESCE(
                (SUM(a.estimated_value) - COALESCE(prev.total_value, 0)) /
                NULLIF(COALESCE(prev.total_value, 0), 0) * 100, 0
            ) as growth_12m_percent
        FROM asset_categories ac
        LEFT JOIN assets a ON ac.id = a.category_id
        LEFT JOIN asset_persona_ownership apo ON a.id = apo.asset_id
        LEFT JOIN ffc_personas fp ON apo.persona_id = fp.persona_id
    )

```



```

LEFT JOIN (
    -- Previous year totals for growth calculation
    SELECT
        category_id,
        SUM(estimated_value) as total_value
    FROM assets a_prev
    WHERE a_prev.updated_at <= (p_report_date - INTERVAL '12 months')
    GROUP BY category_id
) prev ON ac.id = prev.category_id
WHERE (fp.ffc_id = p_ffc_id OR fp.ffc_id IS NULL)
AND (a.status = 'active' OR a.status IS NULL)
GROUP BY ac.id, ac.name, prev.total_value
ORDER BY total_value DESC
)
SELECT * FROM wealth_summary;
END;
$$ LANGUAGE plpgsql;

```

Story 4.3: Comprehensive Audit Trail System

As a compliance officer

I want complete audit trails for all platform activities

So that we maintain regulatory compliance and security accountability

Acceptance Criteria:

- All CRUD operations logged with user context
- IP address and device tracking
- Data change history with before/after values
- Retention policy implementation (7 years)
- Audit log search and filtering
- Suspicious activity detection and alerts

Database Implementation:

```

-- Enhanced audit logging stored procedure
CREATE OR REPLACE FUNCTION log_audit_event(
    p_user_id UUID,
    p_persona_id UUID DEFAULT NULL,
    p_action VARCHAR,
    p_resource_type VARCHAR,
    p_resource_id UUID DEFAULT NULL,
    p_old_values JSONB DEFAULT NULL,
    p_new_values JSONB DEFAULT NULL,
    p_ip_address INET DEFAULT NULL,
    p_user_agent TEXT DEFAULT NULL,
    p_session_id VARCHAR DEFAULT NULL
) RETURNS UUID AS $$
DECLARE
    v_audit_id UUID;
    v_risk_score INTEGER := 0;
BEGIN
    -- Calculate risk score based on action type
    v_risk_score := CASE
        WHEN p_action IN ('delete_asset', 'transfer_ownership', 'change_permissions') THEN 9
        WHEN p_action IN ('create_asset', 'update_asset', 'invite_member') THEN 5
        WHEN p_action IN ('login', 'view_asset', 'search') THEN 1
        ELSE 3
    END;

    -- Increase risk for sensitive operations outside business hours
    IF EXTRACT(hour FROM NOW()) NOT BETWEEN 6 AND 22
        AND p_action IN ('delete_asset', 'transfer_ownership') THEN
        v_risk_score := v_risk_score + 5;
    END IF;

    -- Insert audit record
    INSERT INTO audit_log (
        user_id, persona_id, action, resource_type, resource_id,
        old_values, new_values, ip_address, user_agent, session_id,
        risk_score, created_at
    ) VALUES (
        p_user_id, p_persona_id, p_action, p_resource_type, p_resource_id,
        p_old_values, p_new_values, p_ip_address, p_user_agent, p_session_id,
        v_risk_score, NOW()
    ) RETURNING id INTO v_audit_id;

    -- Trigger alert for high-risk activities

```

```

IF v_risk_score >= 10 THEN
    INSERT INTO security_alerts (
        alert_type, user_id, audit_log_id, severity, message, created_at
    ) VALUES (
        'high_risk_activity', p_user_id, v_audit_id, 'high',
        format('High-risk activity detected: %s by user %s', p_action, p_user_id),
        NOW()
    );
END IF;

RETURN v_audit_id;
END;
$$ LANGUAGE plpgsql;

```

Story 4.4: Bulk Operations & Data Management

As an FFC owner with many assets

I want bulk operations for efficient management

So that I can update multiple assets quickly and maintain data consistency

Acceptance Criteria:

- Bulk asset updates (category, permissions, values)
- Bulk document uploads with batch processing
- Data validation and error reporting
- Rollback capabilities for failed operations
- Progress tracking for long-running operations
- Bulk export and import functionality

Database Implementation:

```

-- Bulk asset update stored procedure
CREATE OR REPLACE FUNCTION bulk_update_assets(
    p_user_id UUID,
    p_asset_updates JSONB
) RETURNS TABLE(
    operation_id UUID,
    total_assets INTEGER,
    successful_updates INTEGER,
    failed_updates INTEGER,
    error_details JSONB
) AS $$
DECLARE
    v_operation_id UUID;
    v_update_record JSONB;
    v_asset_id UUID;
    v_success_count INTEGER := 0;
    v_error_count INTEGER := 0;
    v_errors JSONB := '[]'::JSONB;
    v_total_count INTEGER;
BEGIN
    -- Generate operation ID
    v_operation_id := gen_random_uuid();
    v_total_count := jsonb_array_length(p_asset_updates);

    -- Process each asset update
    FOR v_update_record IN SELECT * FROM jsonb_array_elements(p_asset_updates)
    LOOP
        BEGIN
            v_asset_id := (v_update_record->>'asset_id')::UUID;

            -- Verify user has permission to update this asset
            IF NOT EXISTS (
                SELECT 1 FROM assets a
                JOIN asset_persona_ownership apo ON a.id = apo.asset_id
                JOIN personas p ON apo.persona_id = p.id
                WHERE a.id = v_asset_id AND p.user_id = p_user_id
            ) THEN
                RAISE EXCEPTION 'No permission to update asset %', v_asset_id;
            END IF;

            -- Perform the update
            UPDATE assets SET
                name = COALESCE((v_update_record->>'name')::VARCHAR, name),

```

```

        description = COALESCE((v_update_record->>'description')::TEXT, description),
        estimated_value = COALESCE((v_update_record->>'estimated_value')::DECIMAL, estim
        updated_at = NOW()
    WHERE id = v_asset_id;

    -- Log the change
    PERFORM log_audit_event(
        p_user_id, NULL, 'bulk_update_asset', 'asset', v_asset_id,
        NULL, v_update_record, NULL, NULL, v_operation_id::VARCHAR
    );

    v_success_count := v_success_count + 1;

EXCEPTION WHEN OTHERS THEN
    v_error_count := v_error_count + 1;
    v_errors := v_errors || jsonb_build_object(
        'asset_id', v_asset_id,
        'error', SQLERRM
    );
END;
END LOOP;

RETURN QUERY SELECT
    v_operation_id,
    v_total_count,
    v_success_count,
    v_error_count,
    v_errors;
END;
$$ LANGUAGE plpgsql;

```

Story 4.5: Third-Party Integration Framework

As a platform administrator

I want extensible integration capabilities

So that we can connect with financial institutions and estate planning tools

Acceptance Criteria:

- Generic webhook system for external notifications
- OAuth 2.0 framework for secure API access
- Rate limiting and API key management
- Integration health monitoring

- Data synchronization job management
- Error handling and retry logic

Database Implementation:

```

-- Integration management stored procedure
CREATE OR REPLACE FUNCTION manage_integration(
    p_integration_name VARCHAR,
    p_action VARCHAR, -- 'create', 'update', 'activate', 'deactivate'
    p_config JSONB DEFAULT NULL,
    p_credentials JSONB DEFAULT NULL
) RETURNS TABLE(
    integration_id UUID,
    status VARCHAR,
    message VARCHAR
) AS $$
DECLARE
    v_integration_id UUID;
    v_existing_integration UUID;
BEGIN
    -- Check for existing integration
    SELECT id INTO v_existing_integration
    FROM third_party_integrations
    WHERE name = p_integration_name;

    CASE p_action
        WHEN 'create' THEN
            IF v_existing_integration IS NOT NULL THEN
                RETURN QUERY SELECT v_existing_integration, 'error'::VARCHAR, 'Integration already exists';
                RETURN;
            END IF;

            INSERT INTO third_party_integrations (
                name, config, encrypted_credentials, status, created_at
            ) VALUES (
                p_integration_name, p_config, p_credentials, 'inactive', NOW()
            ) RETURNING id INTO v_integration_id;

            RETURN QUERY SELECT v_integration_id, 'created'::VARCHAR, 'Integration created successfully';

        WHEN 'activate' THEN
            IF v_existing_integration IS NULL THEN
                RETURN QUERY SELECT NULL::UUID, 'error'::VARCHAR, 'Integration not found'::VARCHAR;
                RETURN;
            END IF;

            UPDATE third_party_integrations
            SET status = 'active', activated_at = NOW()

```

```

WHERE id = v_existing_integration;

RETURN QUERY SELECT v_existing_integration, 'activated'::VARCHAR, 'Integration acti

WHEN 'deactivate' THEN
UPDATE third_party_integrations
SET status = 'inactive', deactivated_at = NOW()
WHERE id = v_existing_integration;

RETURN QUERY SELECT v_existing_integration, 'deactivated'::VARCHAR, 'Integration dei

ELSE
RETURN QUERY SELECT NULL::UUID, 'error'::VARCHAR, 'Invalid action'::VARCHAR;
END CASE;
END;
$$ LANGUAGE plpgsql;

```

Story 4.6: Quillt API Integration for Financial Accounts

As an FFC member

I want to connect my bank accounts through Quillt

So that my account balances update automatically without manual entry

Acceptance Criteria:

- Quillt Profile creation and management for FFC members
- Bank account connection flow with OAuth authentication
- Automatic balance refresh (weekly/monthly scheduled)
- Multiple accounts per connection support
- Webhook handling for balance updates
- Connection health monitoring and re-authentication prompts
- Support for 10,000+ financial institutions via Quillt

Integration Architecture:

FFC Member → Quillt Profile → Bank Connection → Multiple Accounts → Forward Assets

Database Implementation:


```

-- Quillt integration management stored procedure
CREATE OR REPLACE FUNCTION manage_quillt_connection(
    p_persona_id UUID,
    p_action VARCHAR, -- 'create_profile', 'connect_bank', 'refresh_balances', 'sync_accounts'
    p_quillt_profile_id VARCHAR DEFAULT NULL,
    p_connection_data JSONB DEFAULT NULL
) RETURNS TABLE(
    operation_id UUID,
    quillt_profile_id VARCHAR,
    status VARCHAR,
    accounts_synced INTEGER,
    message VARCHAR
) AS $$
DECLARE
    v_operation_id UUID;
    v_profile_id VARCHAR;
    v_existing_integration UUID;
    v_account_count INTEGER := 0;
BEGIN
    v_operation_id := gen_random_uuid();

    -- Check for existing Quillt integration
    SELECT quillt_integration_id INTO v_existing_integration
    FROM quillt_integrations
    WHERE persona_id = p_persona_id AND status = 'active';

    CASE p_action
        WHEN 'create_profile' THEN
            -- Create Quillt profile mapping
            INSERT INTO quillt_integrations (
                persona_id, quillt_profile_id, status, created_at
            ) VALUES (
                p_persona_id, p_quillt_profile_id, 'active', NOW()
            );

            v_profile_id := p_quillt_profile_id;

            RETURN QUERY SELECT v_operation_id, v_profile_id, 'profile_created'::VARCHAR, 0, 'Quillt profile created successfully';

        WHEN 'sync_accounts' THEN
            IF v_existing_integration IS NULL THEN
                RETURN QUERY SELECT v_operation_id, NULL::VARCHAR, 'error'::VARCHAR, 0, 'No Quillt integration found';
            ELSE
                RETURN;
            END IF;
    END CASE;
END;

```

```

END IF;

-- Process each account from Quillt connection
FOR v_account_record IN SELECT * FROM jsonb_array_elements(p_connection_data->'accounts')
LOOP
    -- Create or update financial account asset
    INSERT INTO assets (
        category_id, created_by_persona_id, status, external_id, external_provider
    ) VALUES (
        (SELECT id FROM asset_categories WHERE name = 'Financial Accounts'),
        p_persona_id, 'active',
        v_account_record->>'accountId', 'quillt'
    ) ON CONFLICT (external_id, external_provider)
    DO UPDATE SET updated_at = NOW();

    -- Update financial account details
    INSERT INTO financial_accounts (
        asset_id, account_type, institution_name, account_number_masked,
        current_balance, available_balance, last_synced_at
    ) VALUES (
        (SELECT id FROM assets WHERE external_id = v_account_record->>'accountId'),
        v_account_record->>'type',
        v_account_record->>'institutionName',
        v_account_record->>'maskedAccountNumber',
        (v_account_record->>'currentBalance')::DECIMAL,
        (v_account_record->>'availableBalance')::DECIMAL,
        NOW()
    ) ON CONFLICT (asset_id)
    DO UPDATE SET
        current_balance = (v_account_record->>'currentBalance')::DECIMAL,
        available_balance = (v_account_record->>'availableBalance')::DECIMAL,
        last_synced_at = NOW();

    -- Create ownership record
    INSERT INTO asset_persona_ownership (
        asset_id, persona_id, ownership_percentage, ownership_type
    ) VALUES (
        (SELECT id FROM assets WHERE external_id = v_account_record->>'accountId'),
        p_persona_id, 100.00, 'direct'
    ) ON CONFLICT (asset_id, persona_id) DO NOTHING;

    v_account_count := v_account_count + 1;
END LOOP;

```

```

-- Update last sync timestamp
UPDATE quillt_integrations
SET last_synced_at = NOW(), account_count = v_account_count
WHERE persona_id = p_persona_id;

RETURN QUERY SELECT v_operation_id,
    (SELECT quillt_profile_id FROM quillt_integrations WHERE persona_id = p_persona_
    'accounts_synced'::VARCHAR, v_account_count,
    format('%s accounts synced successfully', v_account_count)::VARCHAR;

WHEN 'refresh_balances' THEN
    -- Update balance refresh request timestamp
    UPDATE quillt_integrations
    SET balance_refresh_requested_at = NOW()
    WHERE persona_id = p_persona_id;

    RETURN QUERY SELECT v_operation_id, p_quillt_profile_id, 'refresh_requested'::VARCHAR,

ELSE
    RETURN QUERY SELECT v_operation_id, NULL::VARCHAR, 'error'::VARCHAR, 0, 'Invalid acti
END CASE;
END;
$$ LANGUAGE plpgsql;

-- Quillt webhook processing stored procedure
CREATE OR REPLACE FUNCTION process_quillt_webhook(
    p_event_type VARCHAR,
    p_profile_id VARCHAR,
    p_account_id VARCHAR DEFAULT NULL,
    p_balance_data JSONB DEFAULT NULL,
    p_connection_data JSONB DEFAULT NULL
) RETURNS TABLE(
    webhook_id UUID,
    processed BOOLEAN,
    assets_updated INTEGER,
    message VARCHAR
) AS $$
DECLARE
    v_webhook_id UUID;
    v_persona_id UUID;
    v_asset_id UUID;
    v_updates_count INTEGER := 0;

```

BEGIN

```
v_webhook_id := gen_random_uuid();
```

```
-- Get persona from Quillt profile
```

```
SELECT persona_id INTO v_persona_id
```

```
FROM quillt_integrations
```

```
WHERE quillt_profile_id = p_profile_id AND status = 'active';
```

```
IF v_persona_id IS NULL THEN
```

```
    RETURN QUERY SELECT v_webhook_id, FALSE, 0, 'Profile not found in Forward system'::VARCHAR
```

```
    RETURN;
```

```
END IF;
```

```
-- Log webhook receipt
```

```
INSERT INTO quillt_webhook_log (
```

```
    id, event_type, profile_id, account_id, payload, processed_at
```

```
) VALUES (
```

```
    v_webhook_id, p_event_type, p_profile_id, p_account_id,
```

```
    jsonb_build_object('balance_data', p_balance_data, 'connection_data', p_connection_data,
```

```
    NOW())
```

```
);
```

```
CASE p_event_type
```

```
    WHEN 'balance.created' THEN
```

```
        -- Update specific account balance
```

```
        SELECT id INTO v_asset_id
```

```
        FROM assets
```

```
        WHERE external_id = p_account_id AND external_provider = 'quillt';
```

```
        IF v_asset_id IS NOT NULL THEN
```

```
            UPDATE financial_accounts SET
```

```
                current_balance = (p_balance_data->>'current')::DECIMAL,
```

```
                available_balance = (p_balance_data->>'available')::DECIMAL,
```

```
                credit_limit = (p_balance_data->>'limit')::DECIMAL,
```

```
                last_synced_at = (p_balance_data->>'at')::TIMESTAMP WITH TIME ZONE,
```

```
                updated_at = NOW()
```

```
            WHERE asset_id = v_asset_id;
```

```
            v_updates_count := 1;
```

```
        END IF;
```

```
    WHEN 'profile.ready' THEN
```

```
        -- Trigger full account sync for this profile
```

```

    PERFORM manage_quillt_connection(
        v_persona_id, 'sync_accounts', p_profile_id, p_connection_data
    );

    v_updates_count := (p_connection_data->>'accountCount')::INTEGER;

    WHEN 'connection.disconnected' THEN
        -- Mark integration as requiring re-authentication
        UPDATE quillt_integrations
        SET status = 'disconnected', disconnected_at = NOW()
        WHERE quillt_profile_id = p_profile_id;

    WHEN 'connection.synced.errorred.repairable' THEN
        -- Mark as needing re-authentication
        UPDATE quillt_integrations
        SET status = 'needs_reauth', last_error_at = NOW()
        WHERE quillt_profile_id = p_profile_id;

    END CASE;

    RETURN QUERY SELECT v_webhook_id, TRUE, v_updates_count,
        format('Processed %s event successfully', p_event_type)::VARCHAR;
END;
$$ LANGUAGE plpgsql;

```

Quillt Integration Workflow:

1. **Profile Creation:** Map FFC member to Quillt Profile with metadata
2. **Bank Connection:** User connects via Quillt's OAuth flow
3. **Account Discovery:** Quillt discovers multiple accounts per connection
4. **Asset Creation:** Each account becomes a Financial Account asset
5. **Balance Sync:** Webhook updates or scheduled refreshes update balances
6. **Error Handling:** Connection health monitoring triggers re-authentication

Supported Account Types:

- Checking and Savings Accounts
- Credit Cards and Lines of Credit
- Investment Accounts (401k, IRA, Brokerage)
- Loan Accounts (Mortgage, Auto, Personal)

Story 4.7: Real Estate Data Provider Integration

As an FFC member with real estate assets

I want automated property value updates

So that my real estate portfolio reflects current market values

Acceptance Criteria:

- Research and select optimal real estate data provider (Zillow API, CoreLogic, RentSpree, etc.)
- Property identification via address lookup
- Automated valuation model (AVM) integration
- Comparable sales (comps) data integration
- Property detail enrichment (square footage, lot size, etc.)
- Market trend analysis and alerts
- Scheduled value updates (monthly/quarterly)

Provider Evaluation Criteria:

- **Coverage:** National property database completeness
- **Accuracy:** AVM accuracy vs actual sales data
- **API Quality:** Rate limits, uptime, data freshness
- **Cost Structure:** Per-call pricing vs subscription models
- **Data Richness:** Property details, history, market trends

Database Implementation (Placeholder - will be updated after provider selection):

```

-- Real estate integration framework (provider-agnostic)
CREATE OR REPLACE FUNCTION manage_real_estate_integration(
    p_asset_id UUID,
    p_action VARCHAR, -- 'lookup_property', 'update_valuation', 'sync_details'
    p_address_data JSONB DEFAULT NULL,
    p_provider_data JSONB DEFAULT NULL
) RETURNS TABLE(
    integration_id UUID,
    estimated_value DECIMAL,
    confidence_score DECIMAL,
    last_updated TIMESTAMP WITH TIME ZONE,
    status VARCHAR
) AS $$
DECLARE
    v_integration_id UUID;
    v_property_data JSONB;
BEGIN
    -- Implementation will depend on selected provider
    -- Framework supports multiple providers: Zillow, CoreLogic, etc.

    CASE p_action
        WHEN 'lookup_property' THEN
            -- Address normalization and property lookup
            -- Provider-specific API call implementation
            NULL;

        WHEN 'update_valuation' THEN
            -- Automated valuation model update
            -- Historical value tracking
            NULL;

        WHEN 'sync_details' THEN
            -- Property detail synchronization
            -- Square footage, lot size, year built, etc.
            NULL;
    END CASE;

    RETURN QUERY SELECT
        gen_random_uuid(), 0::DECIMAL, 0::DECIMAL,
        NOW(), 'provider_selection_pending'::VARCHAR;
END;
$$ LANGUAGE plpgsql;

```

Provider Research Tasks:

1. **API Comparison:** Zillow API vs CoreLogic vs RentSpree vs others
2. **Pricing Analysis:** Cost per property lookup and ongoing updates
3. **Data Quality Assessment:** Accuracy testing with known property values
4. **Integration Complexity:** Authentication, rate limits, webhook support
5. **Legal Compliance:** Terms of service and data usage restrictions

Story 4.8: Performance Optimization & Caching

As a system user

I want fast response times even with large datasets

So that the platform remains responsive as our family wealth grows

Acceptance Criteria:

- Redis caching for frequently accessed data
- Database query optimization and indexing
- API response time monitoring (<200ms p95)
- Pagination for large result sets
- Background job processing for heavy operations
- CDN integration for static assets

Development Phases

Phase 1A: Foundation and Core Features (Days 1-60)

Primary Objective: Launch with marketing foundation, secure family onboarding, and comprehensive asset management.

Month 1 (Days 1-30): Infrastructure and Marketing Foundation

Development Priorities:

- **Technical Foundation**
 - Development, staging, and production environments
 - Core database schema implementation
 - CI/CD pipelines with automated testing
 - AWS infrastructure setup (S3, CloudFront, RDS)

- Authentication and authorization systems
- Monitoring and logging infrastructure
- **Builder.io Marketing Platform**
 - [Builder.io](#) integration and configuration
 - Landing page templates and components
 - A/B testing framework setup
 - Analytics and tracking implementation
 - SEO optimization and performance tuning
 - Marketing team training and handoff
- **Core Backend Services**
 - User management and authentication APIs
 - FFC creation and management endpoints
 - SMS/Email service integrations (Twilio, SendGrid)
 - Permission system implementation
 - Audit logging system
 - Database optimization and indexing

Month 2 (Days 31-60): Complete Onboarding and Asset Management

Development Priorities:

- **Secure Onboarding System**
 - User registration and email verification
 - FFC creation wizard
 - Dual-channel invitation system
 - Mandatory phone verification
 - Owner approval workflow
 - Role assignment and management
- **Comprehensive Asset Management**
 - All 13 asset categories implementation
 - Asset-persona ownership system
 - Individual asset permissions
 - Category-specific forms and validation
 - File upload and document management
 - Asset dashboard and overview
- **HEI Integration**
 - API integration for loan data
 - Read-only display in asset system
 - Data synchronization and updates

- Integration with asset permission system

Phase 1A Success Criteria:

- Marketing team publishing content independently
- 90%+ user registration completion rate
- Dual-channel verification >90% success rate
- All 13 asset categories functional
- Asset permission system working correctly
- Zero security breaches
- Landing page conversion rate >2%

Phase 1B: Enhancement and Optimization (Days 61-120)

Primary Objective: Optimize user experience, add advanced features, and prepare for scale.

Development Priorities:

- Advanced search and filtering
- Bulk asset operations
- Enhanced mobile experience
- Performance optimization
- Security enhancements (2FA)
- Advanced analytics and reporting
- User feedback integration
- International expansion features

Conclusion

This Product Requirements Document V1.2 provides a comprehensive foundation for building the Forward Inheritance SaaS platform with a focus on marketing independence, secure family onboarding, and comprehensive asset management. The refined Phase 1A approach balances immediate market needs with long-term strategic vision, ensuring rapid time-to-market while maintaining technical excellence and user security.

The innovative family-first collaboration features combined with individual asset control and enhanced security positioning Forward to capture significant market share in the estate planning and wealth

transfer space. Success depends on flawless execution of the marketing foundation and secure onboarding flow while building toward comprehensive asset management capabilities.

Document Version: 1.2

Last Updated: December 30, 2024

Next Review: January 15, 2025

Changes from V1.1: Added [Builder.io](#) marketing foundation, enhanced security with dual-channel verification, clarified asset permission model, removed chat system from MVP