Priority Features - Detailed TODO Lists & Development Plan

🎯 12 Priority Features Development Tasks

Based strictly on features listed in `docs/priority_features.md`

📅 PHASE 1: Foundation & Quick Wins (Months 1-3)

1. Subscription Manager ⭐ [START FIRST - QUICK WIN]

Track and optimize recurring subscriptions with cancellation reminders

Backend Development Tasks:

- [ ] Create `subscriptions` table with fields:
  - `id`, `user_id`, `name`, `category`, `amount`, `billing_frequency`, `next_billing_date`
  - `provider`, `status`, `auto_renew`, `cancellation_date`, `notes`
- [ ] Create `subscription_categories` table (streaming, utilities, software, etc.)
- [ ] Implement subscription detection algorithm from expense patterns
- [ ] Build `/api/subscriptions` CRUD endpoints
- [ ] Create `/api/subscriptions/detect` endpoint for pattern recognition
- [ ] Implement `/api/subscriptions/analytics` for cost analysis
- [ ] Add subscription reminder system with email notifications
- [ ] Create subscription cost optimization engine
- [ ] Build duplicate subscription detection logic
- [ ] Write comprehensive unit tests for all subscription logic

Frontend Development Tasks:

- [ ] Create `SubscriptionManager.jsx` main component
- [ ] Design subscription overview dashboard with:
  - Monthly/yearly cost summaries
  - Category breakdown charts
  - Upcoming renewal calendar
- [ ] Implement subscription creation/editing forms
- [ ] Add subscription detection interface showing potential subscriptions
- [ ] Create subscription analytics dashboard
- [ ] Implement cancellation reminder system
- [ ] Add subscription sharing/family account features
- [ ] Design mobile-responsive subscription cards
- [ ] Create subscription cost comparison tools
- [ ] Add export functionality for subscription data
- [ ] Write component tests for all subscription features

Integration Tasks:

- [ ] Connect subscription detection to existing expense data
- [ ] Integrate with calendar API for renewal reminders
- [ ] Add email notification system integration
- [ ] Connect to SMS notification system (optional)
- [ ] Create subscription budget impact analysis

Estimated Time: 2-3 weeks

---

2. Travel Budget Planner ⭐ [QUICK WIN]

Trip cost estimation, currency conversion, travel expense tracking

Backend Development Tasks:

- [ ] Create `travel_budgets` table with fields:
  - `id`, `user_id`, `trip_name`, `destination`, `start_date`, `end_date`
  - `total_budget`, `currency`, `status`, `actual_spent`
- [ ] Create `travel_expenses` table linked to travel budgets
- [ ] Create `travel_categories` table (accommodation, food, transport, activities, etc.)
- [ ] Implement currency conversion API integration (Exchange Rate API or Fixer.io)
- [ ] Build `/api/travel-budgets` CRUD endpoints
- [ ] Create `/api/travel-budgets/{id}/expenses` expense tracking endpoints
- [ ] Implement `/api/currencies` for real-time exchange rates
- [ ] Add travel budget analytics and reporting endpoints
- [ ] Create travel expense categorization system
- [ ] Build budget vs actual spending comparison logic
- [ ] Write unit tests for currency conversion and budget calculations

Frontend Development Tasks:

- [ ] Create `TravelBudgetPlanner.jsx` main component
- [ ] Design travel budget creation wizard with:
  - Destination selection with currency auto-detection
  - Budget category templates by destination type
  - Duration-based budget estimation
- [ ] Implement multi-currency expense tracking interface
- [ ] Add real-time currency conversion display
- [ ] Create travel expense entry forms with category selection
- [ ] Design travel budget dashboard with progress visualization
- [ ] Implement budget vs actual spending comparison charts
- [ ] Add travel itinerary integration (optional)
- [ ] Create travel expense receipt upload functionality
- [ ] Design shared travel budget features for group travel
- [ ] Add travel budget export and sharing features
- [ ] Write comprehensive component tests

Integration Tasks:

- [ ] Integrate with currency exchange rate APIs
- [ ] Connect travel expenses to main expense tracking system
- [ ] Add geolocation services for automatic expense categorization
- [ ] Integrate with receipt scanner (when available in Phase 2)

Estimated Time: 2-3 weeks

---

3. Fee Tracker ⭐ [BUILDS ON SUBSCRIPTION MANAGER]

Monitor and minimize banking and investment fees

Backend Development Tasks:

- [ ] Create `fee_categories` table with predefined fee types:
  - Banking fees (overdraft, ATM, monthly maintenance, wire transfer)
  - Investment fees (management, trading, advisory, expense ratios)
  - Credit card fees (annual, late payment, foreign transaction)
- [ ] Create `tracked_fees` table linking fees to transactions and accounts
- [ ] Implement fee detection algorithm from transaction descriptions
- [ ] Build `/api/fees` endpoints for fee tracking and analysis
- [ ] Create `/api/fees/analysis` for fee optimization recommendations
- [ ] Implement fee alert system for unusual or high fees
- [ ] Add fee comparison tools for different financial institutions
- [ ] Create fee forecasting based on usage patterns
- [ ] Build fee minimization suggestion engine
- [ ] Write unit tests for fee detection and analysis algorithms

Frontend Development Tasks:

- [ ] Create `FeeTracker.jsx` main component
- [ ] Design fee dashboard with:
  - Monthly/yearly fee summaries by category
  - Fee trend analysis charts
  - Institution comparison tables
- [ ] Implement fee detection review interface
- [ ] Add manual fee entry and categorization forms
- [ ] Create fee alert configuration dashboard
- [ ] Design fee optimization recommendations display
- [ ] Implement fee comparison tools between institutions
- [ ] Add fee impact calculator (annual cost of fees)
- [ ] Create fee reduction progress tracking
- [ ] Design fee negotiation tips and templates
- [ ] Write component tests for fee tracking features

Integration Tasks:

- [ ] Connect fee detection to existing transaction data
- [ ] Integrate with subscription manager for recurring fees
- [ ] Link to bank account information for institution comparison
- [ ] Connect to investment accounts for investment fee tracking

Estimated Time: 3-4 weeks

---

4. Loan Comparison Tool

Compare mortgage, auto, and personal loan options

Backend Development Tasks:

- [ ] Create `loan_products` table with fields:
  - `id`, `institution_name`, `loan_type`, `interest_rate`, `term_months`
  - `minimum_amount`, `maximum_amount`, `fees`, `requirements`
- [ ] Create `loan_comparisons` table for user-saved comparisons
- [ ] Implement loan calculation engine (monthly payments, total interest, APR)
- [ ] Build `/api/loans` endpoints for loan product management
- [ ] Create `/api/loans/calculate` for payment calculations
- [ ] Implement `/api/loans/compare` for side-by-side comparisons
- [ ] Add loan rate API integration (if available)
- [ ] Create loan affordability calculator
- [ ] Implement amortization schedule generation
- [ ] Build loan pre-qualification assessment tools
- [ ] Write unit tests for all loan calculations

Frontend Development Tasks:

- [ ] Create `LoanComparison.jsx` main component
- [ ] Design loan comparison wizard with:
  - Loan type selection (mortgage, auto, personal)
  - Amount and term input forms
  - Credit score range selection
- [ ] Implement loan calculator with real-time updates
- [ ] Add side-by-side loan comparison table
- [ ] Create amortization schedule visualization
- [ ] Design loan affordability assessment dashboard
- [ ] Implement loan shopping checklist and tips
- [ ] Add loan application tracking system
- [ ] Create loan rate alert system for favorable rates
- [ ] Design pre-qualification form integration
- [ ] Write component tests for loan comparison features

Integration Tasks:

- [ ] Integrate with loan rate APIs or web scraping services

- [ ] Connect to credit score data (if available)
- [ ] Link to existing budget data for affordability analysis
- [ ] Integrate with document vault for loan document storage

Estimated Time: 3-4 weeks

📅 PHASE 2: Document & Security Infrastructure (Months 4-6)

5. Document Vault ⭐ [SECURITY FOUNDATION]

Secure storage for financial documents with encryption

Backend Development Tasks:

- [ ] Set up secure file storage system (AWS S3 or similar with encryption)
- [ ] Create `documents` table with fields:
  - `id`, `user_id`, `filename`, `original_name`, `file_type`, `file_size`
  - `storage_path`, `encryption_key_id`, `category`, `tags`, `upload_date`
- [ ] Create `document_categories` table (tax docs, insurance, loans, etc.)
- [ ] Implement file encryption/decryption system with AES-256
- [ ] Build `/api/documents` CRUD endpoints with security
- [ ] Create `/api/documents/upload` with virus scanning
- [ ] Implement `/api/documents/search` with full-text search
- [ ] Add document versioning and backup system
- [ ] Create document access logging and audit trails
- [ ] Implement document sharing with permission controls
- [ ] Build document expiration and retention policies
- [ ] Write security-focused unit tests

Frontend Development Tasks:

- [ ] Create `DocumentVault.jsx` main component
- [ ] Design secure document upload interface with:
  - Drag-and-drop file upload
  - File type validation and virus scanning feedback
  - Upload progress and encryption status
- [ ] Implement document organization system with folders/tags
- [ ] Add document preview functionality (PDF, images)
- [ ] Create document search and filtering interface
- [ ] Design document sharing and permission management
- [ ] Implement document download with decryption
- [ ] Add document metadata editing forms
- [ ] Create document audit log viewer
- [ ] Design mobile document scanner integration
- [ ] Write security-focused component tests

Security & Infrastructure Tasks:

- [ ] Implement end-to-end encryption for document storage
- [ ] Set up automated backup and disaster recovery
- [ ] Create document access control and permissions system
- [ ] Implement secure document deletion (cryptographic shredding)
- [ ] Add document integrity verification (checksums)
- [ ] Set up virus scanning and malware detection
- [ ] Create audit logging for all document operations
- [ ] Implement document retention policy enforcement

Estimated Time: 4-5 weeks

---

6. Receipt Scanner ⭐ [BUILDS ON DOCUMENT VAULT]

OCR technology to scan and digitize receipts

Backend Development Tasks:

- [ ] Create `receipts` table with fields:
  - `id`, `user_id`, `document_id`, `merchant_name`, `total_amount`, `date`
  - `currency`, `tax_amount`, `payment_method`, `ocr_confidence`, `manual_verified`
- [ ] Create `receipt_line_items` table for itemized receipts
- [ ] Integrate OCR service (Google Vision API, AWS Textract, or Tesseract)
- [ ] Build `/api/receipts/scan` endpoint for OCR processing
- [ ] Implement receipt data extraction algorithm
- [ ] Create `/api/receipts/verify` for manual verification
- [ ] Add receipt-to-expense conversion system
- [ ] Implement receipt duplicate detection
- [ ] Build receipt search and filtering endpoints
- [ ] Create receipt analytics and reporting
- [ ] Write OCR accuracy tests and benchmarks

Frontend Development Tasks:

- [ ] Create `ReceiptScanner.jsx` main component
- [ ] Design mobile-first receipt capture interface with:
  - Camera integration for photo capture
  - Image preview and crop functionality
  - Real-time OCR processing feedback
- [ ] Implement receipt data verification interface
- [ ] Add receipt-to-expense conversion workflow
- [ ] Create receipt gallery and management system
- [ ] Design receipt data editing forms
- [ ] Implement receipt search and filtering
- [ ] Add batch receipt processing capabilities
- [ ] Create receipt accuracy improvement feedback system
- [ ] Design receipt analytics dashboard
- [ ] Write component tests for OCR workflow

Integration Tasks:

- [ ] Integrate with Document Vault for receipt storage
- [ ] Connect receipt data to expense tracking system
- [ ] Link receipt scanner to travel budget expenses
- [ ] Integrate with mobile camera APIs
- [ ] Connect to merchant database for better recognition

Estimated Time: 5-6 weeks

---

7. Estate Planning

Will and beneficiary management

Backend Development Tasks:

- [ ] Create `estate_documents` table with fields:
  - `id`, `user_id`, `document_type`, `status`, `last_updated`
  - `beneficiaries`, `executor_info`, `legal_review_date`
- [ ] Create `beneficiaries` table with contact and allocation information
- [ ] Create `estate_assets` table linking assets to estate planning
- [ ] Build `/api/estate-planning` CRUD endpoints
- [ ] Implement will template system with legal compliance
- [ ] Create beneficiary notification system
- [ ] Add estate document versioning and history
- [ ] Implement estate value calculation from existing assets
- [ ] Build estate planning checklist and progress tracking
- [ ] Create emergency contact management system
- [ ] Write tests for legal compliance requirements

Frontend Development Tasks:

- [ ] Create `EstatePlanning.jsx` main component
- [ ] Design estate planning wizard with:
  - Asset inventory from existing data
  - Beneficiary management forms
  - Will template selection and customization
- [ ] Implement beneficiary allocation interface
- [ ] Add estate document management dashboard
- [ ] Create estate planning progress tracking
- [ ] Design emergency contact management
- [ ] Implement estate planning reminders and updates
- [ ] Add legal resource links and guidance
- [ ] Create estate planning checklist interface
- [ ] Design estate document sharing (with executor)
- [ ] Write component tests for estate planning workflow

Legal & Compliance Tasks:

- [ ] Research legal requirements by state/jurisdiction
- [ ] Create compliant will templates
- [ ] Implement legal disclaimer and guidance system
- [ ] Add attorney referral system integration
- [ ] Create legal document review reminders

Estimated Time: 4-5 weeks

---

8. Financial Power of Attorney

Emergency contact and account access management

Backend Development Tasks:

- [ ] Create `power_of_attorney` table with fields:
  - `id`, `user_id`, `attorney_contact_id`, `authorization_level`, `effective_date`
  - `expiration_date`, `status`, `authorization_document_id`
- [ ] Create `emergency_contacts` table with detailed contact information
- [ ] Create `account_authorizations` table for specific account access
- [ ] Build `/api/power-of-attorney` management endpoints
- [ ] Implement authorization verification system
- [ ] Create emergency access request workflow
- [ ] Add authorization audit logging system
- [ ] Implement authorization revocation system
- [ ] Build emergency contact notification system
- [ ] Create authorization document management
- [ ] Write security-focused tests for authorization system

Frontend Development Tasks:

- [ ] Create `PowerOfAttorney.jsx` main component
- [ ] Design power of attorney setup wizard with:
  - Emergency contact selection and verification
  - Authorization level configuration
  - Legal document upload and verification
- [ ] Implement authorization management dashboard
- [ ] Add emergency contact management interface
- [ ] Create authorization audit log viewer
- [ ] Design emergency access request system
- [ ] Implement authorization revocation interface
- [ ] Add legal guidance and template system
- [ ] Create authorization status monitoring
- [ ] Write component tests for authorization workflow

Security & Legal Tasks:

- [ ] Implement multi-factor authentication for attorney access
- [ ] Create secure authorization verification system
- [ ] Add legal document verification requirements
- [ ] Implement emergency access protocols
- [ ] Create authorization abuse detection system

Estimated Time: 4-5 weeks

📅 PHASE 3: Advanced Integrations (Months 7-12)

9. Email Receipt Processing ⭐ [BUILDS ON RECEIPT SCANNER]

Automatically process emailed receipts

Backend Development Tasks:

- [ ] Set up email integration system (IMAP/Exchange integration)
- [ ] Create `email_accounts` table for user email configuration
- [ ] Create `email_receipts` table for processed email receipts
- [ ] Implement email parsing and attachment extraction
- [ ] Build receipt detection algorithm for email content
- [ ] Create `/api/email-receipts` processing endpoints
- [ ] Implement email security and authentication verification
- [ ] Add email rule management system
- [ ] Create email processing queue and batch system
- [ ] Implement spam and phishing detection for receipts
- [ ] Build email receipt forwarding system
- [ ] Write tests for email processing security

Frontend Development Tasks:

- [ ] Create `EmailReceiptProcessor.jsx` main component
- [ ] Design email account configuration interface
- [ ] Implement email receipt processing dashboard
- [ ] Add email rule management interface
- [ ] Create processed receipt review workflow
- [ ] Design email receipt search and filtering
- [ ] Implement email processing status monitoring
- [ ] Add email receipt accuracy improvement tools
- [ ] Create email security and privacy settings
- [ ] Write component tests for email processing workflow

Security & Integration Tasks:

- [ ] Implement OAuth2 for email account access
- [ ] Add email encryption and secure storage
- [ ] Create email access audit logging

- [ ] Integrate with existing receipt scanner infrastructure
- [ ] Implement email processing rate limiting

Estimated Time: 6-8 weeks

---

10. Family Financial Dashboard ⭐ [MAJOR FEATURE]

Multi-user accounts with permission levels

Backend Development Tasks:

- [ ] Design multi-tenant database architecture
- [ ] Create `family_groups` table with group management
- [ ] Create `family_members` table with role-based permissions
- [ ] Create `permission_levels` table (admin, manager, viewer, child)
- [ ] Implement data isolation and security for family accounts
- [ ] Build `/api/family` group management endpoints
- [ ] Create `/api/family/members` member management endpoints
- [ ] Implement permission-based data access controls
- [ ] Add family data aggregation and reporting
- [ ] Create family budget and goal sharing system
- [ ] Implement family expense approval workflows
- [ ] Build family communication and notification system
- [ ] Write comprehensive multi-user security tests

Frontend Development Tasks:

- [ ] Create `FamilyDashboard.jsx` main component
- [ ] Design family group creation and management interface
- [ ] Implement role-based permission management
- [ ] Add family member invitation and onboarding system
- [ ] Create family financial overview dashboard
- [ ] Design shared budget and goal management
- [ ] Implement family expense tracking and approval
- [ ] Add family communication center
- [ ] Create family financial education center
- [ ] Design family privacy and data sharing controls
- [ ] Write component tests for multi-user functionality

Multi-User Architecture Tasks:

- [ ] Implement role-based access control (RBAC)
- [ ] Create data privacy and sharing controls
- [ ] Add family data aggregation algorithms
- [ ] Implement cross-user notification system
- [ ] Create family account billing and subscription management

Estimated Time: 8-10 weeks

---

11. Bank Account Integration ⭐ [MOST COMPLEX]

Real-time bank feeds and automatic transaction import

Backend Development Tasks:

- [ ] Research and select bank integration provider (Plaid, Yodlee, Finicity)
- [ ] Create `bank_connections` table for account linking
- [ ] Create `external_accounts` table for bank account information
- [ ] Create `imported_transactions` table with deduplication
- [ ] Implement Plaid API integration for account linking
- [ ] Build `/api/bank-integration` connection management endpoints
- [ ] Create automatic transaction import and categorization
- [ ] Implement transaction deduplication and matching
- [ ] Add real-time webhook processing for account updates
- [ ] Create account balance synchronization system
- [ ] Implement bank integration error handling and retry logic
- [ ] Build comprehensive integration monitoring and logging
- [ ] Write security and compliance tests

Frontend Development Tasks:

- [ ] Create `BankIntegration.jsx` main component
- [ ] Design secure bank account linking interface
- [ ] Implement Plaid Link integration for account connection
- [ ] Add bank account management dashboard
- [ ] Create transaction import review and approval system
- [ ] Design automatic categorization settings
- [ ] Implement bank account synchronization monitoring
- [ ] Add bank integration troubleshooting tools
- [ ] Create bank account security and privacy settings
- [ ] Write component tests for bank integration workflow

Security & Compliance Tasks:

- [ ] Implement bank-level security requirements
- [ ] Add comprehensive audit logging for all bank operations
- [ ] Create data encryption for bank credentials
- [ ] Implement PCI DSS compliance requirements
- [ ] Add bank integration monitoring and alerting

Estimated Time: 10-12 weeks

---

12. Financial Institution APIs ⭐ [FINAL INTEGRATION]

Direct integration with major banks and brokerages

Backend Development Tasks:

- [ ] Extend bank integration to support investment accounts
- [ ] Create `brokerage_connections` table for investment account linking
- [ ] Create `investment_transactions` table for trading activity
- [ ] Implement multiple API provider support (Plaid, Yodlee, Finicity)
- [ ] Build comprehensive account aggregation system
- [ ] Create `/api/financial-institutions` management endpoints
- [ ] Implement investment position and performance tracking
- [ ] Add multi-institution portfolio aggregation
- [ ] Create financial institution data synchronization
- [ ] Implement advanced transaction categorization across institutions
- [ ] Build financial institution API monitoring and failover
- [ ] Write comprehensive integration tests

Frontend Development Tasks:

- [ ] Create `FinancialInstitutions.jsx` management interface
- [ ] Design comprehensive account aggregation dashboard
- [ ] Implement multi-institution portfolio view
- [ ] Add institution connection management tools
- [ ] Create aggregated financial reporting dashboard
- [ ] Design institution-specific data handling interfaces
- [ ] Implement advanced filtering and search across institutions
- [ ] Add institution API status monitoring dashboard
- [ ] Create comprehensive financial institution settings
- [ ] Write component tests for multi-institution functionality

Advanced Integration Tasks:

- [ ] Implement real-time data synchronization across all institutions
- [ ] Create advanced financial analytics across all accounts
- [ ] Add institution-specific feature support
- [ ] Implement comprehensive error handling and user notification
- [ ] Create advanced security monitoring for all institution connections

Estimated Time: 12-14 weeks

📋 Implementation Timeline Summary

Phase 1 Duration: 10-14 weeks

- Subscription Manager: 2-3 weeks
- Travel Budget Planner: 2-3 weeks
- Fee Tracker: 3-4 weeks
- Loan Comparison Tool: 3-4 weeks

Phase 2 Duration: 17-21 weeks

- Document Vault: 4-5 weeks
- Receipt Scanner: 5-6 weeks
- Estate Planning: 4-5 weeks
- Financial Power of Attorney: 4-5 weeks

Phase 3 Duration: 36-44 weeks

- Email Receipt Processing: 6-8 weeks
- Family Financial Dashboard: 8-10 weeks
- Bank Account Integration: 10-12 weeks
- Financial Institution APIs: 12-14 weeks

Total Project Duration: 63-79 weeks (15-19 months)

🎯 Resource Requirements

Team Composition Recommendations:

- **1 Senior Full-Stack Developer**: Architecture and complex integrations
- **2 Frontend Developers**: UI/UX implementation and testing
- **2 Backend Developers**: API development and database design
- **1 Security Specialist**: Security implementation and compliance
- **1 DevOps Engineer**: Infrastructure and deployment
- **1 QA Engineer**: Testing and quality assurance

Critical Success Factors:

1. **Security First**: All features must meet banking-level security standards
2. **Compliance**: Ensure regulatory compliance for financial integrations
3. **User Experience**: Maintain intuitive interface despite feature complexity
4. **Performance**: Handle large volumes of financial data efficiently
5. **Reliability**: Ensure 99.9% uptime for critical financial features

---

Last Updated: August 2025
Priority Features TODO Lists v1.0
12 Features from docs/priority_features.md