PolicyPal Technical Documentation

System Architecture Overview

PolicyPal is a modern web application built with React, TypeScript, and Firebase, designed to help users manage their financial policies, investments, and documents.

Table of Contents

Technology Stack

Frontend

- **Framework**: React 18.x with TypeScript
- **Build Tool**: Vite
- **Routing**: React Router v6
- **UI Components**: Custom components with Radix UI primitives
- **Styling**: Tailwind CSS with custom design system
- **State Management**: React Hooks (useState, useEffect, custom hooks)
- **Forms**: Controlled components with validation
- **Icons**: Lucide React
- **Date Handling**: date-fns
- **Notifications**: Sonner (toast notifications)
- **PDF Generation**: jsPDF with jspdf-autotable

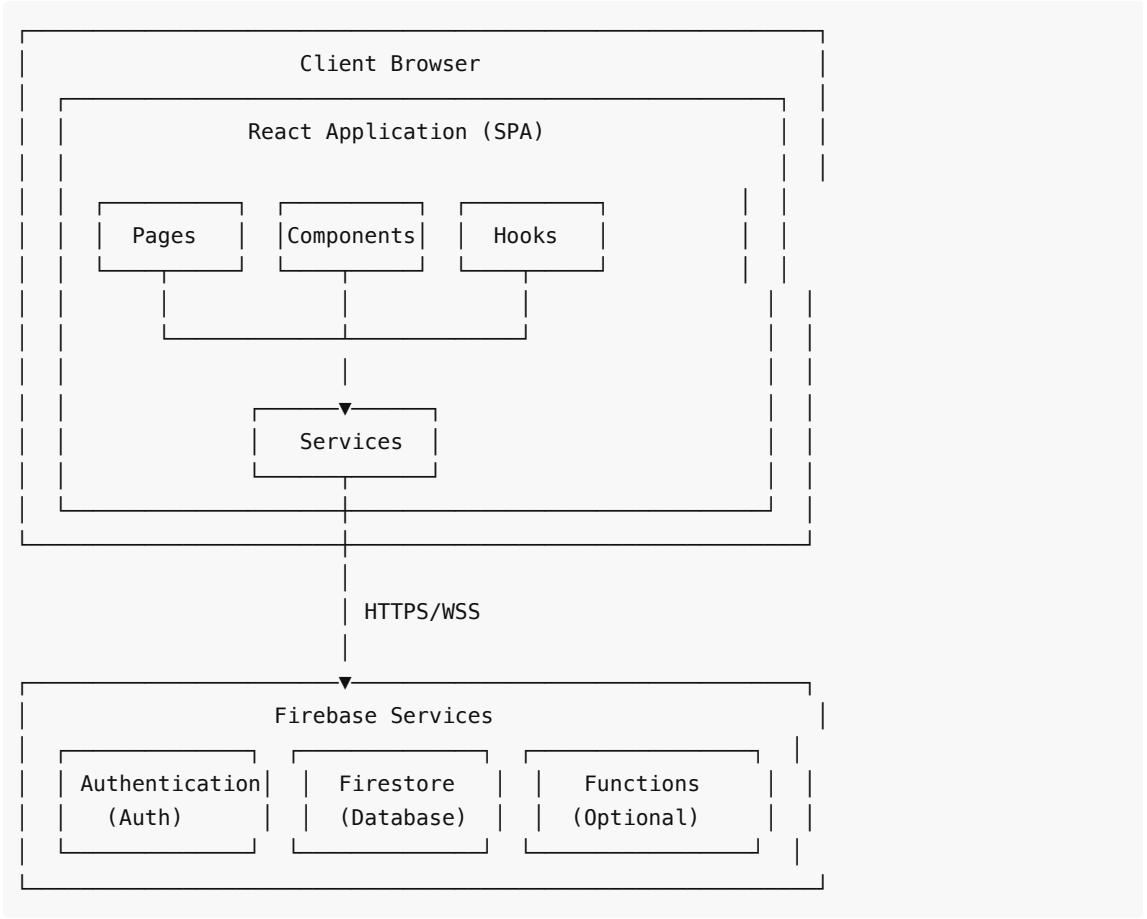
Backend & Services

- **Authentication**: Firebase Authentication
- **Database**: Cloud Firestore (NoSQL)
- **Storage**: Base64 encoding in Firestore (no Firebase Storage due to plan limitations)
- **Hosting**: Firebase Hosting (recommended)

Development Tools

- **Package Manager**: npm
- **TypeScript**: v5.x
- **Linting**: ESLint
- **Version Control**: Git

System Architecture



Architecture Layers

1. **Presentation Layer** (React Components)
   - Pages: Route-level components
   - Components: Reusable UI elements
   - Layout: Header, Sidebar, common layouts
2. **Business Logic Layer** (Custom Hooks)
   - useFinancialData: Manages policies, investments, alerts
   - useDocuments: Handles document operations
   - useMoneyVisibility: Privacy toggle
   - useTheme: Theme management
   - useGlobalSearch: Search functionality
3. **Data Layer** (Firebase)
   - Firestore: Real-time database
   - Authentication: User management
   - Security Rules: Access control

Database Schema

Collections Structure

```
firestore/
├── users/
│   └── {userId}/
│       └── (user metadata - optional)
│
├── policies/
│   └── {policyId}/
│       ├── id: string
│       ├── userId: string
│       ├── name: string
│       ├── provider: string
│       ├── policyNumber: string
│       ├── type: 'life' | 'car' | 'home' | 'medical' | 'other'
│       ├── coverage: number
│       ├── premium: number
│       ├── premiumFrequency: 'monthly' | 'annual'
│       ├── startDate: string (ISO)
│       ├── expiryDate: string (ISO)
│       └── createdAt: string (ISO)
│
├── investments/
│   └── {investmentId}/
│       ├── id: string
│       ├── userId: string
│       ├── name: string
│       ├── provider: string
│       ├── type: 'stocks' | 'bonds' | 'mutual-funds' | 'crypto' | 'real-estate' |
'other'
│       ├── initialValue: number
│       ├── currentValue: number
│       ├── returnPercentage: number
│       ├── status: 'growing' | 'stable' | 'declining'
│       ├── purchaseDate: string (ISO)
│       └── createdAt: string (ISO)
│
├── alerts/
│   └── {alertId}/
│       ├── id: string
│       ├── userId: string
│       ├── type: 'renewal' | 'payment' | 'review' | 'custom'
│       ├── title: string
│       ├── description: string
│       ├── dueDate: string (ISO)
│       ├── priority: 'low' | 'medium' | 'high'
│       ├── relatedItemId: string (optional)
│       ├── relatedItemType: 'policy' | 'investment' (optional)
│       └── createdAt: string (ISO)
```

```
│
├── documents/
│   └── {documentId}/
│       ├── id: string
│       ├── userId: string
│       ├── name: string
│       ├── type: 'personal' | 'policy' | 'investment' | 'tax' | 'statement' | 'other'
│       ├── category: string
│       ├── fileData: string (base64)
│       ├── fileType: string (MIME type)
│       ├── size: string
│       ├── uploadDate: string (ISO)
│       └── createdAt: string (ISO)
│
└── profiles/
    └── {userId}/
        ├── displayName: string
        ├── phone: string
        ├── address: string
        ├── bio: string
        └── updatedAt: string (ISO)
```

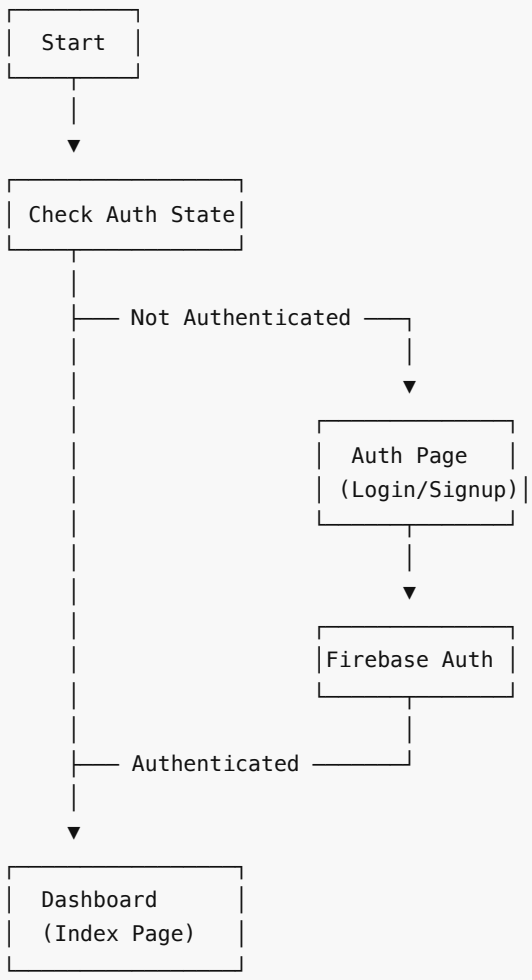
Firestore Security Rules

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Helper function to check authentication
    function isAuthenticated() {
      return request.auth != null;
    }

    // Helper function to check ownership
    function isOwner(userId) {
      return request.auth.uid == userId;
    }

    // Policies collection
    match /policies/{policyId} {
      allow read, write: if isAuthenticated() && isOwner(resource.data.userId);
      allow create: if isAuthenticated() && isOwner(request.resource.data.userId);
    }

    // Investments collection
    match /investments/{investmentId} {
      allow read, write: if isAuthenticated() && isOwner(resource.data.userId);
      allow create: if isAuthenticated() && isOwner(request.resource.data.userId);
    }

    // Alerts collection
    match /alerts/{alertId} {
```

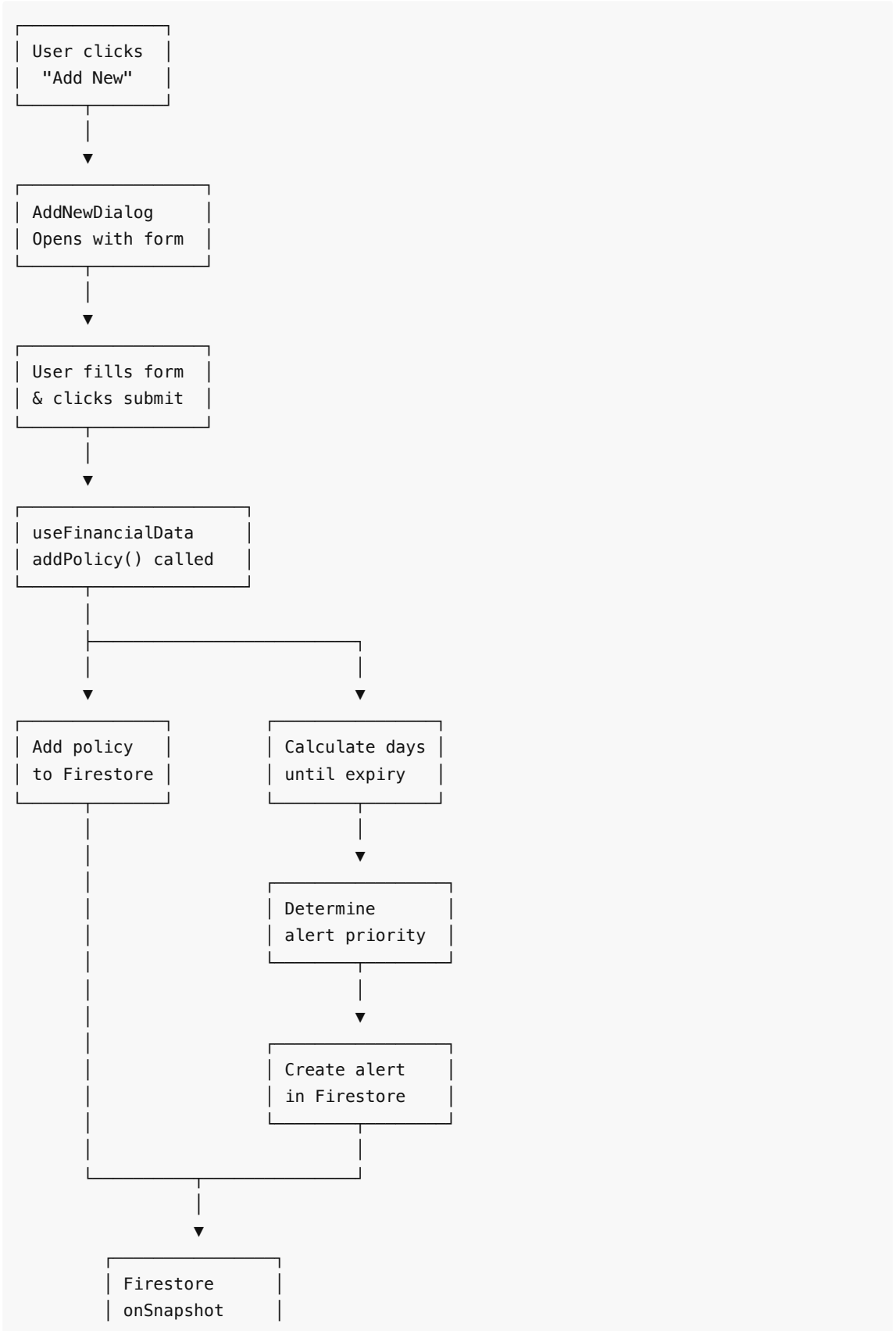
```
      allow read, write: if isAuthenticated() && isOwner(resource.data.userId);
      allow create: if isAuthenticated() && isOwner(request.resource.data.userId);
    }

    // Documents collection
    match /documents/{documentId} {
      allow read, write: if isAuthenticated() && isOwner(resource.data.userId);
      allow create: if isAuthenticated() && isOwner(request.resource.data.userId);
    }

    // Profiles collection
    match /profiles/{userId} {
      allow read, write: if isAuthenticated() && isOwner(userId);
    }
  }
}
```

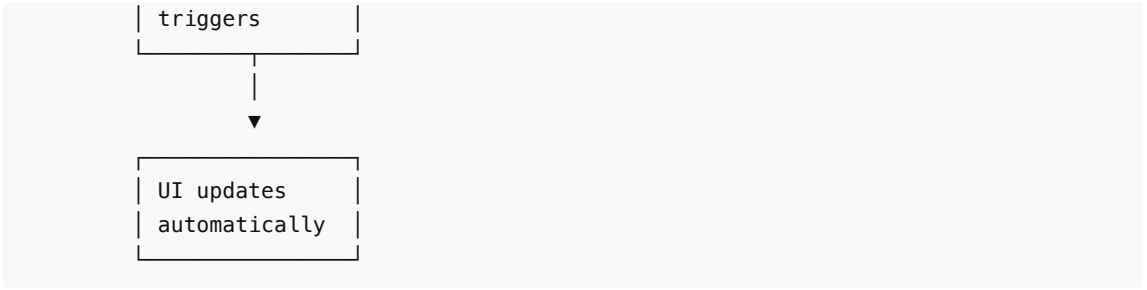
Application Flow

User Authentication Flow

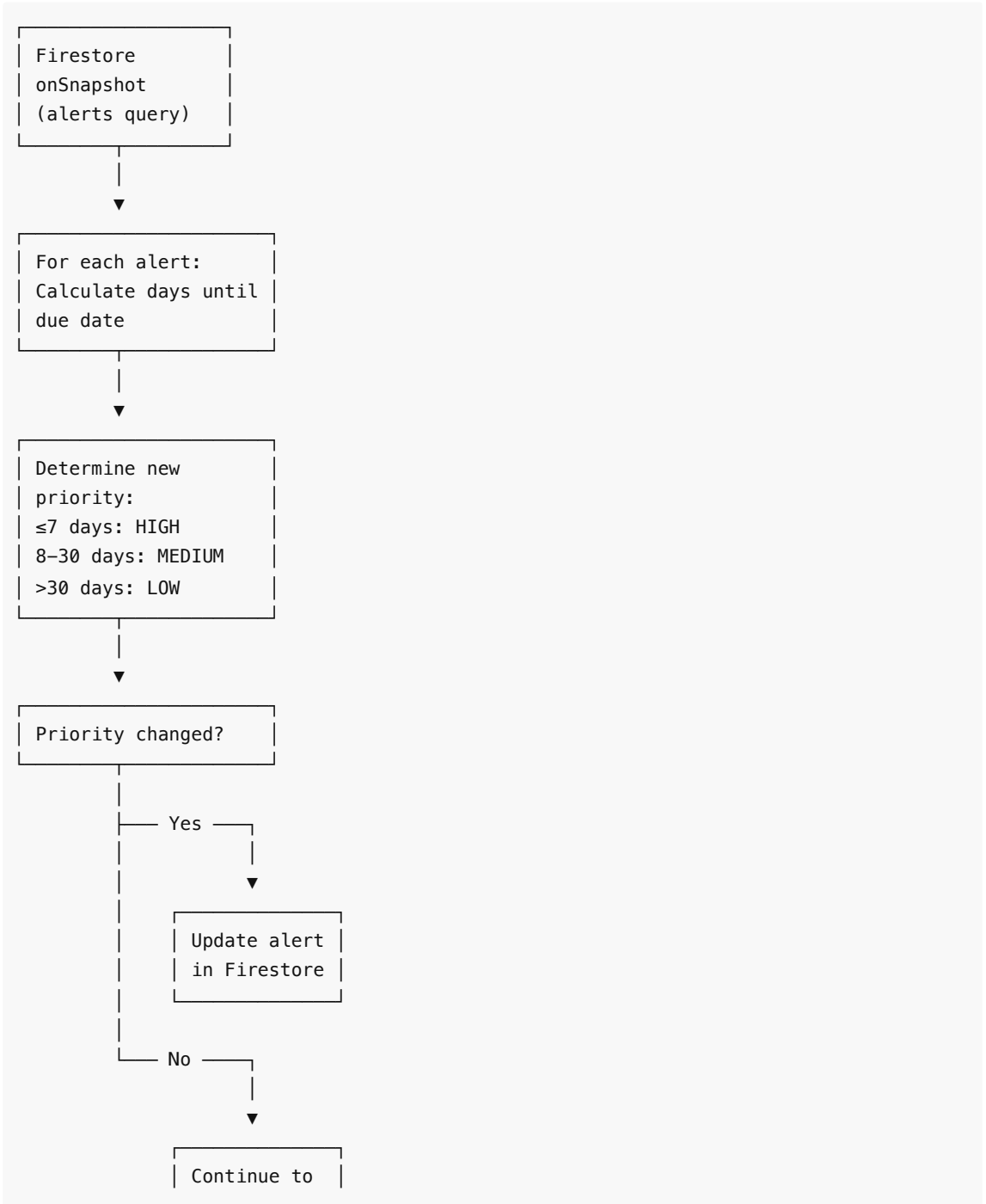


Data Flow - Adding a Policy





Alert Priority Update Flow



```
              | next alert   |
              └──────────────┘
```

Component Architecture

Component Hierarchy

```
App
├── BrowserRouter
│   ├── Routes
│   │   ├── Index (Main Dashboard)
│   │   │   ├── Header
│   │   │   ├── Sidebar
│   │   │   ├── Tutorial
│   │   │   ├── AddNewDialog
│   │   │   └── Content (based on activeTab)
│   │   │       ├── Dashboard View
│   │   │       │   ├── QuickStats
│   │   │       │   ├── StatCard (x4)
│   │   │       │   ├── AlertCard (x3)
│   │   │       │   ├── PortfolioChart
│   │   │       │   ├── PolicyCard (x2)
│   │   │       │   └── InvestmentCard (x2)
│   │   │       ├── Policies View
│   │   │       │   └── PolicyCard (multiple)
│   │   │       ├── Investments View
│   │   │       │   └── InvestmentCard (multiple)
│   │   │       ├── Alerts View (AlertsPage)
│   │   │       ├── Documents View (DocumentsPage)
│   │   │       ├── Profile View (ProfilePage)
│   │   │       └── Settings View (SettingsPage)
│   │   ├── Auth (Login/Signup)
│   │   └── NotFound (404)
│   └── Providers
│       ├── QueryClientProvider
│       ├── TooltipProvider
│       └── MoneyVisibilityProvider
```

Key Components

Pages

- **Index.tsx**: Main application container with routing logic
- **Auth.tsx**: Authentication page (login/signup)
- **Alerts.tsx**: Alerts management page
- **Documents.tsx**: Document storage and management
- **Profile.tsx**: User profile management
- **Settings.tsx**: Application settings
- **Notfound.tsx**: 404 error page

Layout Components

- **Header.tsx**: Top navigation bar with search, theme toggle, visibility toggle, notifications, profile
- **Sidebar.tsx**: Left navigation menu with tabs and "Add New" button

Dashboard Components

- **QuickStats.tsx**: Four-card summary of key metrics
- **StatCard.tsx**: Individual stat display card
- **PolicyCard.tsx**: Policy information card
- **InvestmentCard.tsx**: Investment information card
- **AlertCard.tsx**: Alert/reminder card
- **PortfolioChart.tsx**: Investment portfolio visualization

Dialog Components

- **AddNewDialog.tsx**: Modal for adding policies, investments, or alerts

Tutorial Component

- **Tutorial.tsx**: Interactive onboarding tutorial with step-by-step guidance

---

State Management

Custom Hooks

useFinancialData

Purpose: Manages all financial data (policies, investments, alerts)

State:

```
{
  policies: Policy[]
  investments: Investment[]
  alerts: Alert[]
  loading: boolean
}
```

Methods:

- `addPolicy(policy)` : Add new policy and auto-create alert
- `addInvestment(investment)` : Add new investment
- `addAlert(alert)` : Add new alert
- `updateAlert(id, alert)` : Update existing alert
- `deleteAlert(id)` : Delete alert

Real-time Updates: Uses Firestore onSnapshot for live data synchronization

useDocuments

Purpose: Manages document uploads and storage

State:

```
{
  documents: Document[]
}
```

Methods:

- `addDocument(name, type, category, fileData)` : Upload new document
- `deleteDocument(id)` : Delete document

useMoneyVisibility

Purpose: Privacy feature to hide/show financial amounts

State:

```
{
  isVisible: boolean
}
```

Methods:

- `toggle()` : Toggle visibility
- `formatCurrency(amount, isVisible)` : Format currency with visibility

Persistence: Stores preference in localStorage

useTheme

Purpose: Dark/Light mode management

State:

```
{
  isDark: boolean
}
```

Methods:

- `toggle()` : Switch between dark and light mode

Persistence: Stores preference in localStorage and applies CSS class to document root

useGlobalSearch

Purpose: Search across policies, investments, and alerts

Parameters:

- `query: string`
- `policies: Policy[]`
- `investments: Investment[]`
- `alerts: Alert[]`

Returns:

```
{
  policies: Policy[]
  investments: Investment[]
```

```
  alerts: Alert[]
}
```

Performance: Uses useMemo for optimized filtering

Authentication & Security

Firebase Authentication

Supported Methods:

- Email/Password authentication

Auth Flow:

1. User enters credentials
2. Firebase Authentication validates
3. JWT token issued
4. Token stored in browser
5. Token included in all Firestore requests
6. Firestore security rules validate token

Security Features

1. **Authentication Required**: All data operations require valid authentication
2. **User Isolation**: Security rules ensure users can only access their own data
3. **HTTPS Only**: All communication encrypted in transit
4. **XSS Protection**: React's built-in XSS protection
5. **CSRF Protection**: Firebase handles CSRF tokens
6. **Input Validation**: Client-side validation before submission
7. **Privacy Toggle**: Money visibility feature for public viewing

Data Privacy

- **No Third-Party Sharing**: User data never shared
- **Encrypted Storage**: Firestore encrypts data at rest
- **User Control**: Users can delete their data anytime
- **Base64 Storage**: Documents stored as base64 in Firestore (no separate storage service)

Key Features Implementation

1. Auto-Alert Creation with Dynamic Priority

Implementation: `useFinancialData.ts` - `addPolicy()`

```
// Calculate days until expiry
const daysUntilExpiry = Math.floor(
  (expiryDate.getTime() - new Date().getTime()) / (1000 * 60 * 60 * 24)
);

// Determine priority
let priority: 'low' | 'medium' | 'high';
if (daysUntilExpiry <= 30) {
```

```
  priority = 'high';
} else if (daysUntilExpiry <= 60) {
  priority = 'medium';
} else {
  priority = 'low';
}
```

Auto-Update: Alert priorities automatically update when alerts are loaded:

```
// In onSnapshot callback
data.forEach(async (alert) => {
  const daysUntilDue = calculateDaysUntil(alert.dueDate);
  const newPriority = determinePriority(daysUntilDue);

  if (alert.priority !== newPriority) {
    await updateDoc(doc(db, 'alerts', alert.id), { priority: newPriority });
  }
});
```

2. Global Search

Implementation: `useGlobalSearch.ts`

```
export function useGlobalSearch(query, policies, investments, alerts) {
  return useMemo(() => {
    if (!query.trim()) return { policies, investments, alerts };

    const lowerQuery = query.toLowerCase();

    return {
      policies: policies.filter(p =>
        p.name.toLowerCase().includes(lowerQuery) ||
        p.provider.toLowerCase().includes(lowerQuery) ||
        p.policyNumber.toLowerCase().includes(lowerQuery)
      ),
      investments: investments.filter(i =>
        i.name.toLowerCase().includes(lowerQuery) ||
        i.provider.toLowerCase().includes(lowerQuery)
      ),
      alerts: alerts.filter(a =>
        a.title.toLowerCase().includes(lowerQuery) ||
        a.description.toLowerCase().includes(lowerQuery)
      )
    };
  }, [query, policies, investments, alerts]);
}
```

3. PDF Export

Implementation: `lib/exportPDF.ts`

Uses jsPDF and jspdf-autotable to generate comprehensive financial reports including:

- User information
- Policy summary table
- Investment summary table
- Alert summary table
- Total statistics

4. Interactive Tutorial

Implementation: `components/tutorial/Tutorial.tsx`

Features:

- 12-step guided tour
- Highlights UI elements with pulsing animation
- Positioned tooltips that stay within viewport
- Skip/Next navigation
- Completion tracking in localStorage
- Can be restarted from Settings

Step Targeting: Uses `data-tutorial` attributes on elements

5. Document Upload with Base64 Storage

Implementation: `pages/Documents.tsx`

```
const reader = new FileReader();
reader.onload = async (e) => {
  const fileData = e.target?.result as string; // base64

  await addDoc(collection(db, 'documents'), {
    name,
    type,
    category,
    fileData, // Store base64 directly
    fileType: file.type,
    size: `${(file.size / 1024).toFixed(0)} KB`,
    userId: user.uid
  });
};
reader.readAsDataURL(file);
```

Why Base64?: Firebase Storage not available on free plan, so documents are stored as base64 strings in Firestore.

API Integration

Firebase SDK Integration

Configuration: `src/integrations/firebase/config.ts`

```
import { initializeApp } from 'firebase/app';
import { getAuth } from 'firebase/auth';
import { getFirestore } from 'firebase/firestore';

const firebaseConfig = {
  apiKey: import.meta.env.VITE_FIREBASE_API_KEY,
  authDomain: import.meta.env.VITE_FIREBASE_AUTH_DOMAIN,
  projectId: import.meta.env.VITE_FIREBASE_PROJECT_ID,
  storageBucket: import.meta.env.VITE_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: import.meta.env.VITE_FIREBASE_MESSAGING_SENDER_ID,
  appId: import.meta.env.VITE_FIREBASE_APP_ID
};

const app = initializeApp(firebaseConfig);
export const auth = getAuth(app);
export const db = getFirestore(app);
```

Environment Variables

Required in `.env`:

```
VITE_FIREBASE_API_KEY=
VITE_FIREBASE_AUTH_DOMAIN=
VITE_FIREBASE_PROJECT_ID=
VITE_FIREBASE_STORAGE_BUCKET=
VITE_FIREBASE_MESSAGING_SENDER_ID=
VITE_FIREBASE_APP_ID=
```

Deployment

Build Process

```
# Install dependencies
npm install

# Build for production
npm run build

# Preview production build
npm run preview
```

Firebase Hosting Deployment

```
# Install Firebase CLI
npm install -g firebase-tools

# Login to Firebase
firebase login
```

```
# Initialize Firebase
firebase init hosting

# Deploy
firebase deploy --only hosting
```

Build Output

- **Output Directory**: `dist/`
- **Entry Point**: `index.html`
- **Assets**: Bundled and optimized by Vite
- **Code Splitting**: Automatic route-based splitting

Performance Optimizations

1. **Code Splitting**: React.lazy for route-based splitting
2. **Tree Shaking**: Vite removes unused code
3. **Minification**: Production builds are minified
4. **Caching**: Static assets cached with content hashing
5. **Lazy Loading**: Images and components loaded on demand
6. **Memoization**: useMemo for expensive computations

Development Setup

Prerequisites

- Node.js 18+ and npm
- Firebase account
- Git

Installation

```
# Clone repository
git clone <repository-url>
cd my-policy-pal

# Install dependencies
npm install

# Create .env file
cp .env.example .env
# Edit .env with your Firebase credentials

# Start development server
npm run dev
```

Development Scripts

```
npm run dev          # Start dev server
npm run build        # Build for production
```

```
npm run preview      # Preview production build
npm run lint         # Run ESLint
```

Testing Strategy

Recommended Testing Approach

1. **Unit Tests**: Test individual hooks and utility functions
2. **Component Tests**: Test React components in isolation
3. **Integration Tests**: Test feature flows
4. **E2E Tests**: Test complete user journeys

Testing Tools (Recommended)

- **Vitest**: Unit testing
- **React Testing Library**: Component testing
- **Playwright**: E2E testing
- **Firebase Emulator**: Local Firebase testing

Monitoring & Analytics

Recommended Tools

1. **Firebase Analytics**: User behavior tracking
2. **Firebase Performance**: Performance monitoring
3. **Sentry**: Error tracking
4. **Google Analytics**: Web analytics

Future Enhancements

Planned Features

1. **Multi-currency Support**: Real-time exchange rates
2. **Recurring Payments**: Track subscription payments
3. **Budget Planning**: Monthly budget tracking
4. **Data Export**: CSV/Excel export
5. **Mobile App**: React Native version
6. **Notifications**: Email/SMS reminders
7. **Collaboration**: Share policies with family
8. **AI Insights**: Smart recommendations
9. **OCR**: Auto-extract data from documents
10. **Backup/Restore**: Data backup functionality

Technical Improvements

1. **Offline Support**: PWA with service workers
2. **Real-time Sync**: Optimistic UI updates
3. **Performance**: Virtual scrolling for large lists
4. **Accessibility**: WCAG 2.1 AA compliance
5. **Internationalization**: Multi-language support
6. **Testing**: Comprehensive test coverage
7. **CI/CD**: Automated deployment pipeline

Troubleshooting

Common Issues

Build Errors:

- Clear node_modules and reinstall
- Check Node.js version (18+)
- Verify all environment variables

Firebase Connection Issues:

- Verify Firebase config in .env
- Check Firebase project settings
- Ensure Firestore is enabled

Authentication Issues:

- Check Firebase Authentication is enabled
- Verify email/password provider is active
- Check security rules

Data Not Loading:

- Check browser console for errors
- Verify Firestore security rules
- Ensure user is authenticated

API Reference

Custom Hooks API

useFinancialData()

```
const {
  policies,      // Policy[]
  investments,   // Investment[]
  alerts,        // Alert[]
  loading,       // boolean
  addPolicy,     // (policy: Omit<Policy, 'id'>) => Promise<void>
  addInvestment, // (investment: Omit<Investment, 'id'>) => Promise<void>
  addAlert,      // (alert: Omit<Alert, 'id'>) => Promise<void>
  updateAlert,   // (id: string, alert: Omit<Alert, 'id'>) => Promise<void>
  deleteAlert    // (id: string) => Promise<void>
} = useFinancialData();
```

useDocuments()

```
const {
  documents,     // Document[]
  addDocument,   // (name, type, category, fileData) => Promise<void>
```

```
  deleteDocument // (id: string) => Promise<void>
} = useDocuments();
```

useMoneyVisibility()

```
const {
  isVisible,     // boolean
  toggle,        // () => void
  formatCurrency // (amount: number, isVisible: boolean) => string
} = useMoneyVisibility();
```

useTheme()

```
const {
  isDark,  // boolean
  toggle   // () => void
} = useTheme();
```

useGlobalSearch()

```
const filtered = useGlobalSearch(
  query: string,
  policies: Policy[],
  investments: Investment[],
  alerts: Alert[]
);
// Returns: { policies: Policy[], investments: Investment[], alerts: Alert[] }
```

Version History

v1.0.0 (Current)

- Initial release
- Core features: Policies, Investments, Alerts, Documents
- Firebase integration
- Auto-alert system with dynamic priorities
- Global search
- PDF export
- Interactive tutorial
- Dark/Light theme
- Money visibility toggle

License

Contact & Support

Developer: PolicyPal Team **Email**: dev@policypal.com **Documentation**: https://docs.policypal.com

---

Last Updated: 2024 **Version**: 1.0.0