Custody Wallet Solution High Level Functional Document

Notice

All information contained in this documentation is proprietary to WadzPay. WadzPay reserves any and all intellectual property rights in respect of this document.

The Customer is given the right to use this document in accordance with the agreement with WadzPay. Unless as permitted by the relevant agreement, no part of this documentation may be reproduced, stored, or transmitted in any form or by electronic, mechanical, recording, or any other means, without the prior written permission of WadzPay.

This document is confidential and proprietary and is not intended for use by any person other than the named Customer.

Version Control

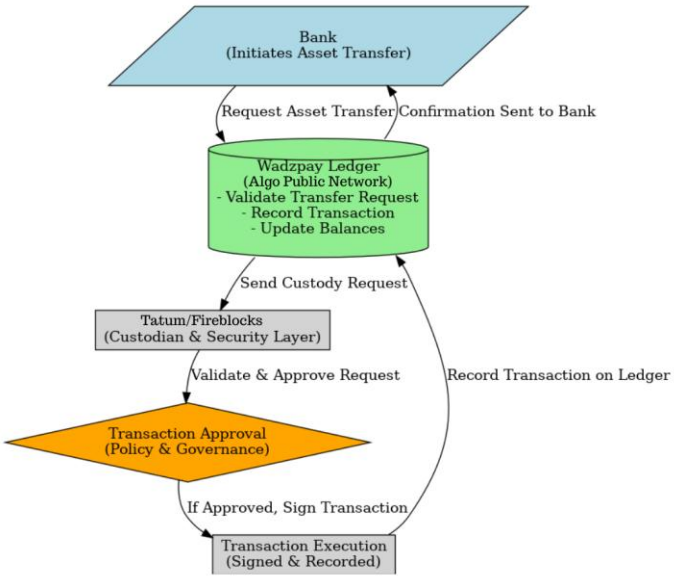
Version	Date	Author	Reviewer	Description
1.0	11-Apr-2025	Vinod L	Ashish K	Baseline

Table of Contents

1. Wadzpay Wallet

Wadzpay Wallet is an open-source, web-based crypto wallet designed for seamless Algorand blockchain asset management. Built with simplicity and security in mind, it enables users to securely create, manage, and transact Algorand assets with ease. Whether you're a developer, a blockchain enthusiast, or an everyday user, Wadzpay Wallet offers a lightweight, intuitive, and real-time experience for handling digital assets.

2. High Level Functional Flow



3. Functionality and Features

- **User Authentication:** Simple email/password-based authentication system
- **Wallet Management:** Generate and recover Algorand wallets using mnemonics
- **Transaction History:** View your transaction history with filtering and pagination
- **Send Tokens:** Transfer Algorand tokens to other wallet addresses
- **Real-time Updates:** Get real-time balance updates via WebSocket connection
- **Secure Storage:** Private keys are securely stored and encrypted
- **Mnemonic Backup:** Back up your wallet with a 24-word mnemonic phrase

Product Access: Algo Safeguard Wallet

Test Faucet: Algorand Testnet Bank
<< Required to fund the wallets during test phases >>

4. Technology Stack

- **Frontend:** React, TypeScript, Tailwind CSS, shadcn-ui
- **Backend:** Supabase for database and authentication
- **Blockchain:** Algorand via Tatum API
- **Build Tool:** Vite

5. Prerequisites

Before setting up this project, you'll need:

1. Node.js (v18 or later) and npm
2. A Supabase account (free tier works fine)
3. A Tatum API key (for Algorand blockchain interactions)

6. Project Setup

6.1 Clone the repository

```
git clone YOUR_REPOSITORY_URL>
cd algorand-web-wallet
```

6.2 Install dependencies

```
npm install
```

6.3 Supabase Setup

1. Create a new Supabase project at https://app.supabase.com/
2. Note your Supabase URL and anon key from the API settings
3. Run the database setup SQL scripts (available in the
   I. Create the customers table
   II. Create the wallets table
   III. Set up the necessary RLS policies
   IV. Create the required database functions

6.4 Configure Supabase Edge Functions

1. Install the Supabase CLI if you haven't already
2. Deploy the Algorand edge function:
3. Supabase functions deploy algorand
4. Set up the required secrets in Supabase:
5. Supabase secrets set TATUM_API_KEY=your_tatum_api_key

6.5 Get a Tatum API Key

1. Register for a Tatum account at https://tatum.io/
2. Create an API key from your dashboard
3. Add this key to your Supabase edge function secrets as shown above

6.6 Environment Configuration

The project uses Supabase client setup which is already configured in the `src/integrations/supabase/client.ts` file. No additional configuration is needed for the frontend to connect to Supabase.

6.7 Run the project locally

```
npm run dev
```

The application will be available at http://localhost:8080/

6.8 Development Workflow

1. **Authentication**: Users can register and login with email and password
2. **Wallet Creation**: First-time users get a new Algorand wallet created automatically
3. **Mnemonic Backup**: Users must backup their wallet mnemonic after registration
4. **Dashboard**: View balance and recent transactions
5. **Send**: Send Algorand tokens to other addresses
6. **Transactions**: View detailed transaction history

6.9 Deploying to Production

You can deploy this project to any static hosting service like Vercel, Netlify, or GitHub Pages. Here's how to deploy to Netlify:

1. Build the project:
2. npm run build
3. Deploy to Netlify:
    - Install Netlify CLI: `npm install -g netlify-cli`
    - Deploy: `netlify deploy --prod --dir=dist`

Alternatively, connect your GitHub repository to Netlify or Vercel for automatic deployments.

6.10 Using a Custom Domain

Setting Up Your Custom Domain:

1. **Purchase a Domain:** Buy a domain from a registrar like Namecheap, GoDaddy, or Google Domains.

2. Connect to Your Hosting Provider:

a. In your hosting dashboard (Netlify, Vercel, etc.), go to the domain settings.
b. Add your custom domain in the custom domains section.
c. Follow the provider's instructions to verify domain ownership.

3. Configure DNS Records:

a. Go to your domain registrar's DNS settings.
b. Add the DNS records recommended by your hosting provider:
   i. For Netlify: Add the Netlify DNS records or use Netlify DNS
   ii. For Vercel: Add the required A records or CNAME records
   iii. For Cloudflare: Add the site to Cloudflare and update nameservers

4. **Wait for DNS Propagation:**

a. DNS changes can take 24-48 hours to fully propagate globally.
b. Use tools like https://dnschecker.org/ to verify propagation.

5. **Set Up SSL Certificate:**

a. Most providers will automatically provision an SSL certificate.
b. Ensure your hosting provider has SSL enabled for your domain.

6.11 License

This project is licensed under the MIT License - see the LICENSE file for details.