Borderless – Design Document

1. System Architecture Overview

The app is designed as a frontend-first, single-page application with modular components, a global state store, and integration with mock and live APIs.

Architecture Flow:

User → UI Components (Vue.js + TailwindCSS) → Pinia Stores → API Layer (Axios + FX API) → Mock Backend / Live FX API

2. Component Structure

The app is built with Vue.js and Tailwind CSS, using Vue Single File Components (SFC) to build reusable and maintainable code. Components are arranged by use case:

Core Components:

- Layouts: Main layout wrapper for pages
- Views: All routable pages
- Base: Reusable utility components (Input, Button, Select, Toast, Modal)
- Table: Render tables in component
- Cards: Unique cards like BalanceCard, WalletCard, TransactionCard
- Composable: Reusable functions callable from any component
- Stores: State management connecting frontend to backend
- Charts: Individual chart components for data visualization

Reasoning:

- Components are modular and reusable
- Separation of concerns improves maintainability and testability
- Base components ensure a consistent design system

3. State Management

Pinia Stores:

- authStore: User data, profile settings, notification preferences
- walletsStore: Wallets, balances, deposit, send, and swap actions
- currenciesStore: Supported currencies
- transactionsStore: Transaction history, pagination
- depositAccountsStore: User’s deposit accounts
- fxStore: Connects to live FX data

Flow:

1. User interacts with UI → triggers an action in a Pinia store
2. Store calls API layer to fetch/update data
3. Store updates reactive state, triggering UI re-render

Reasoning:

- Pinia allows centralized global state with reactive bindings
- Actions and getters separate logic from view, improving maintainability
- Supports lazy-loading and caching for scalability

4. API Interaction

Endpoints:

- Wallet creation: POST /wallets (Mock backend)
- Deposit funds: PATCH /wallets/:id (Mock backend)
- Swap currencies: POST /swap (Mock backend with FX API)
- Send funds: POST /transactions (Mock backend)
- FX Rates: GET /fx-rates (Live API integration / Mock)
- Transaction history: GET /transactions (Pagination)

Error Handling:

- Retry logic for transient network errors
- Fallback to cached FX rates if API fails
- Toast notifications for user-facing errors
- Reusable error component for consistent handling

5. Scalability & Performance

- Lazy Loading: Load non-critical components (charts, transaction history) asynchronously
- Pagination: Avoid rendering large transaction lists at once
- Caching FX Rates: Reduce API calls, improve performance
- Code Splitting: Bundle components per page for faster initial load
- Responsive Design: TailwindCSS ensures mobile-first layouts
- Icons: Bootstrap Icons ensure consistency across the app

6. Testing Strategy

- Unit Tests: Core flows (wallet creation, FX swap, transactions)
- Integration Tests: End-to-end user journey: onboarding → deposit → swap → send
- Coverage Target: ≥80% for critical paths
- Tools: Vitest for unit tests, Cypress for end-to-end testing

7. UX Considerations

- Clear Onboarding: Minimal friction with email/phone-based mock authentication
- Consistent Feedback: Toast messages for success/failure
- Accessible Design: Focus states, ARIA labels, keyboard navigation
- Analytics Visualization: Charts for FX analytics, tables for performance metrics

Conclusion

The frontend-first architecture combined with reactive Pinia stores ensures a scalable, maintainable, and performant cross-border payment dashboard. Modular components, robust error handling, and responsive design guarantee a strong user experience in both sandbox and live deployments.