VAULTA

Fintech Banking Dashboard

Full Project Report

Prepared by: Lokpavan P

GitHub: Vaulta---Fintech-Dashboard

Platform: Vercel (Live Deployment)

Date: February 2026

1. Executive Summary

Vaulta is a full-stack fintech banking dashboard designed to simulate real-world cross-border banking operations for the global workforce. Built as a single-page React application, Vaulta provides users with a comprehensive set of financial tools including fund transfers, currency exchange, transaction history, analytics, and an admin control panel.

The project was developed from scratch using React 18, Context API for state management, and Recharts for data visualisation. It is deployed live on Vercel and connected to a GitHub repository for continuous deployment. The application supports multiple user roles (standard user and admin), real-time balance updates, and a multi-currency system supporting 8 international currencies.

Key Highlights

- Fully functional transactional banking app — send, receive, exchange, and request money
- Role-based authentication — separate dashboards for users and admins
- 8-currency FX exchange engine with live rate calculation and 0.5% fee transparency
- Real-time balance and transaction updates without page refresh
- Deployed on Vercel with GitHub CI/CD pipeline
- 40+ files across a well-structured component architecture

2. Project Overview

2.1 Project Details

Field	Details
Project Name	Vaulta — Fintech Banking Dashboard
Version	v2.0 (Transactional)
Type	Single-Page React Application (SPA)
Framework	React 18 with Create React App
Deployment	Vercel (Auto-deploy from GitHub)
Repository	github.com/pavan1832/Vaulta---Fintech-Dashboard
Total Files	40+ source files

2.2 Problem Statement

Modern cross-border workers — remote employees, freelancers, and digital nomads — face significant friction when managing money across currencies and geographies. Traditional banking apps are either too complex, geographically restricted, or charge hidden fees for international transfers. Vaulta addresses this by providing a clean, transparent, and unified banking interface for the global workforce.

2.3 Objectives

- Build a production-quality fintech dashboard with real transactional capabilities
- Implement role-based access control for users and administrators
- Support multi-currency operations with transparent fee structure
- Create a responsive, professional UI with dark theme design system
- Deploy the application with a CI/CD pipeline for continuous delivery

3. Technology Stack

Layer	Technology	Purpose
UI Framework	React 18	Component-based UI with hooks
State Management	Context API	Auth state + app/transaction state
Charts	Recharts 2.x	Area, Bar, Pie charts for analytics
Styling	Inline CSS + Tokens	Design token system, zero build config
Build Tool	Create React App	Webpack bundling and dev server
Fonts	Google Fonts	DM Sans, DM Serif Display, JetBrains Mono
Version Control	Git + GitHub	Source control and CI/CD trigger
Deployment	Vercel	Auto-deploy on every git push
Auth	JWT Simulation	Session-based auth with sessionStorage

4. Application Architecture

4.1 Project Structure

The application follows a clean separation of concerns with 6 distinct layers:

Directory	Responsibility
src/styles/	Design tokens (colours, fonts) and global CSS animations
src/utils/	Pure utility functions — currency, date, number formatters
src/services/	API service layer and in-memory mock database with FX rates
src/context/	React Context providers — AuthContext and AppContext
src/components/	13 reusable UI primitives — Card, Btn, Modal, Badge, Avatar, etc.
src/pages/	8 full pages + 4 payment modals — Dashboard, Payments, History, Analytics, Admin

4.2 State Management

The application uses two React Context providers instead of Redux, which is appropriate for this scale:

- AuthContext — manages user session, login/logout, JWT token, and the refreshUser() callback for live balance updates after transactions
- AppContext — manages transaction list, pagination, filters, monthly chart data, and the prependTransaction() function for real-time transaction insertion

4.3 Service Layer

All data operations are isolated in apiService.js — no component ever touches the database directly. This means the mock database can be swapped for a real REST API without changing any UI code. Every operation is async with simulated network delay to match real-world behaviour.

4.4 Data Flow

User action (e.g. Send Money) → Modal calls apiService → Service validates and mutates mockDb → Returns result → Context updates balance via refreshUser() and prepends transaction via prependTransaction() → All subscribed components re-render with new data automatically.

5. Features & Functionality

5.1 Authentication System

- Login and signup pages with JWT token simulation
- Session persistence via sessionStorage — stays logged in on page refresh
- Role-based access — user accounts see the banking dashboard, admin accounts see the admin panel
- Demo accounts pre-seeded: user (demo1234) and admin (admin1234)

5.2 Dashboard

- Balance hero card showing live account balance with currency and country
- Quick action buttons — Send Money, Add Funds, Exchange, Request Money
- 4 stat cards — Total Inflow, Total Outflow, Net Savings, Transaction Count
- Area chart showing monthly cash flow (credit vs debit) for last 7 months
- Recent transactions panel with live feed

5.3 Payments Hub

- Dedicated page with 4 action tiles — Send, Add Funds, Exchange, Request
- Live available balance display with green indicator
- FX rate board showing all 7 exchange rates from the user's home currency

5.4 Send Money

- 2-step confirmation flow — form → review screen → success
- Quick-select buttons for existing Borderless users
- Amount presets ($25, $50, $100, $250, $500) for fast entry
- Real-time balance validation — blocks transfer if insufficient funds
- Optional note field for payment description
- Free transfers between Borderless accounts — zero fee

5.5 Add Funds

- Deposit via 6 payment methods — Bank Transfer, UPI, Debit Card, Credit Card, NEFT/RTGS, Wire Transfer
- Amount presets ($500 to $10,000) with manual entry
- Maximum single deposit limit of $50,000
- Instant balance credit with transaction record created

5.6 Currency Exchange

- Supports 8 currencies — USD, EUR, GBP, INR, AED, SGD, CAD, AUD
- Live rate preview as user types the amount
- Transparent 0.5% fee shown before confirmation
- Swap button to quickly reverse the currency direction
- Full conversion details on success screen — rate, fee, received amount

5.7 Request Money

- Generates a shareable deep-link URL with amount and message pre-filled
- One-click copy to clipboard with visual confirmation

5.8 Transaction History

- Paginated table with 10 records per page
- 6 filters — search by description/reference, type (credit/debit), min/max amount, date range
- Skeleton loaders during data fetch, empty state, and error state
- Each row shows date, description, category, reference, amount, type badge, status badge

5.9 Analytics

- Bar chart — monthly credit vs debit volume for last 7 months
- Pie chart — spending breakdown by category
- Category progress bars with percentage and amount for 8 categories

5.10 Admin Panel

- Platform metrics — total users, total volume, total transactions, active today
- System health bars — API uptime, transaction success rate, KYC completion, SLA
- User management table — view all accounts with balance and join date
- Transaction drill-down — click any user to see their full transaction history

6. Component Library

The application includes 13 reusable UI primitives that are used consistently across all pages:

Component	Description
Skeleton	Animated shimmer placeholder shown while data is loading
Spinner	Circular loading indicator for async button states
Card	Surface container with optional glow effect for highlighted sections
Btn	Button with 6 variants (primary, green, gold, danger, ghost, neutral), 3 sizes, loading state
FieldInput	Styled text input with label, helper text, and inline error display
FieldSelect	Styled dropdown select with consistent dark theme styling
Badge	Colour-coded status pill — credit (green), debit (red), completed, pending, failed
Avatar	Circular initials avatar with configurable size and accent colour
StatCard	Metric card with label, value, icon, sub-text, trend indicator, and loading skeleton
AlertBox	Inline alert banner for success and error messages with dismiss button
Modal	Full-screen overlay shell used by all 4 payment modals — click-outside to close
Sidebar	Navigation sidebar with active page indicator, user info, and sign out button
TopBar	Sticky page header with page title, current date, and user avatar

7. Deployment & CI/CD

7.1 Deployment Pipeline

The project uses a GitHub → Vercel continuous deployment pipeline. Every commit pushed to the main branch automatically triggers a new Vercel build and deployment with zero manual intervention required.

#	Step	Details
1	Code change	Developer edits code locally or on GitHub
2	Git push	git add . → git commit → git push to main branch
3	Vercel trigger	GitHub webhook notifies Vercel of new commit
4	Build	Vercel runs npm install && npm run build (react-scripts build)
5	Deploy	Build output (build/ folder) served on global CDN
6	Live	Updated app live within ~60 seconds of git push

7.2 Environment Configuration

A critical environment variable is set in Vercel to prevent ESLint warnings from failing the production build:

- CI = false — prevents Vercel from treating ESLint warnings as build errors

This mirrors the local development behaviour where warnings are displayed but do not block compilation.

7.3 Build Output

- Build command: npm run build
- Output directory: build/
- Node.js version: 22.x
- Build time: approximately 18 seconds
- Bundle size: optimised production build with code splitting

8. Design System

8.1 Colour Palette

Token	Hex Value	Usage
bg	#080C18	Page background — deep navy
surface	#0F1623	Card backgrounds and panels
accent	#00C6FF	Primary action colour — cyan blue
green	#0FE89A	Credit transactions and success states
red	#FF4D6A	Debit transactions and error states
gold	#F5C842	Currency exchange and pending states
purple	#A78BFA	Request money and analytics accent

8.2 Typography

- DM Serif Display — headings and balance figures (elegant, financial feel)
- DM Sans — body text, labels, navigation (clean and readable)
- JetBrains Mono — amounts, reference codes, dates (monospaced for alignment)

8.3 Animations

- fadeUp — page and modal entrance animation
- scaleIn — modal pop-in effect
- shimmer — skeleton loading placeholders
- spin — spinner rotation for loading states

9. Challenges & Solutions

Challenge	Solution
ESLint warnings failing Vercel CI build	Set CI=false in Vercel environment variables to match local behaviour. Also fixed all 7 warnings — unused imports, missing useEffect dependencies.
Real-time balance updates across pages	Implemented refreshUser() in AuthContext that patches sessionStorage and local state simultaneously, ensuring all subscribed components re-render.
Transaction history not updating after payment	Added prependTransaction() to AppContext which inserts new transactions at the top of the list without requiring a full data refetch.
Currency symbol not updating after change	Root cause was sessionStorage caching the old user object. Solution was clearing session storage and replacing all user?.currency references with the target currency code.
Vercel not picking up GitHub changes	Resolved by deleting and recreating the Vercel project with a fresh GitHub connection, eliminating stale cache and webhook issues.

10. Testing Scenarios

The following end-to-end scenarios were used to validate all transactional features:

#	Scenario	Expected Result
1	Login with user credentials	Redirects to Dashboard with balance displayed
2	Login with admin credentials	Redirects to Admin Overview panel
3	Send ₹100 to another user	Both balances update, 2 transactions created, success screen shown
4	Send money with insufficient balance	Error message shown, transaction blocked
5	Add ₹5,000 via UPI	Balance increases, deposit transaction appears in history
6	Exchange INR to USD	Calculates converted amount, deducts amount + 0.5% fee
7	Generate payment request link	Shareable URL generated, copy to clipboard works
8	Filter transactions by date range	Table updates to show only matching records
9	Admin views user transactions	Drill-down panel shows full transaction history for selected user
10	Sign out and sign back in	Session cleared, redirected to login, session restored on re-login

11. Future Improvements

11.1 Backend Integration

- Replace mockDb.js with a real REST API (Node.js + Express or Next.js API routes)
- Integrate PostgreSQL or MongoDB for persistent data storage
- Implement real JWT authentication with refresh tokens

11.2 Features

- Real-time FX rates via an external API (Wise, XE, or Open Exchange Rates)
- Push notifications for incoming transfers
- Bill splitting and group payment features
- Scheduled/recurring transfers
- PDF statement download for transaction history
- Biometric authentication (WebAuthn)

11.3 Technical

- Migrate from Create React App to Vite for faster builds
- Add React Testing Library unit and integration tests
- Implement React Query for server state management and caching
- Add PWA support for mobile installation
- Dark/light theme toggle

12. Conclusion

Vaulta successfully demonstrates a production-quality fintech application built entirely with React and modern frontend architecture patterns. The project goes beyond a read-only dashboard — it implements real transactional flows with proper validation, optimistic UI updates, multi-step confirmation flows, and a comprehensive design system.

The codebase is structured for maintainability with a clean separation between the service layer, state management, reusable components, and page-level logic. The CI/CD pipeline ensures rapid iteration — any change pushed to GitHub is live on the deployed URL within 60 seconds.

The project serves as a strong foundation that can be extended with a real backend, live FX data, and additional financial features to become a fully functional banking product.

Total Pages	Source Files
12	40+