Banking Application Design Document

1. Overview

The Banking Application is a full-stack web solution built with a React frontend and an Express.js backend. It provides basic banking functionality, including user authentication, account management, money transfers, and transaction history tracking. The application is designed to be simple yet extensible, with automated Selenium test generation via generate-tests.js.

Purpose

- Allow users to log in, view accounts, transfer money between accounts (acc123, acc456), and review transaction history.
- Demonstrate integration of a React frontend with an Express backend.
- Enable automated testing with Selenium for functional validation.

Scope

- *Frontend*: React app with routing, state management, and API integration.
- *Backend*: Express.js server with in-memory data storage (expandable to a database).
- *Testing*: Selenium tests generated dynamically using AI and RAG.

---

2. Architecture

High-Level Architecture

[Client Browser] ↔ [React Frontend (Port 3000)] ↔ [Express Backend (Port 5000)] ↑ ↑ [Selenium Tests] [In-Memory Store]

- *React Frontend*: Single-page application (SPA) running on http://localhost:3000.
- *Express Backend*: RESTful API server on http://localhost:5000.
- *In-Memory Store*: Temporary storage for accounts and transactions (replaceable with MongoDB/PostgreSQL).
- *Selenium Tests*: Generated by generate-tests.js, executed against the frontend.

Tech Stack

- *Frontend*:
  - React 18.2.0
  - React Router DOM 6.22.3
  - CSS (basic styling)
- *Backend*:
  - Express.js 4.18.2
  - CORS 2.8.5
- *Testing*:
  - Selenium WebDriver
  - Gemini 2.0 Flash (via @google/generative-ai)
  - @xenova/transformers for embeddings

---

3. Frontend Design

Directory Structure

banking-app/src/ ├── components/ │ ├── Login.js │ ├── TransferForm.js │ ├── AccountList.js │ ├── TransactionHistory.js ├── App.js ├── App.css ├── index.js

Components

1. App.js

- *Purpose*: Root component with routing and authentication state.
- *State*: isAuthenticated (boolean).
- *Routes*:
  - /: Login
  - /accounts: AccountList (protected)
  - /transfer: TransferForm (protected)
  - /history: TransactionHistory (protected)
- *Navigation*: Links to accounts, transfer, history, and logout button.

2. Login.js

- *Purpose*: Handles user authentication.
- *Inputs*: username, password (with data-testid="username", data-testid="password").
- *API Call*: POST /api/login.

- *Output*: Redirects to /accounts on success, shows error on failure (data-testid="error-message").

3. TransferForm.js

- *Purpose*: Facilitates money transfers between accounts.
- *Inputs*: fromAccount, toAccount, amount (with data-testid="fromAccount", data-testid="toAccount", data-testid="amount").
- *API Call*: POST /api/transfer.
- *Output*: Success/error message (data-testid="message").

4. AccountList.js

- *Purpose*: Displays all accounts.
- *API Call*: GET /api/accounts.
- *Output*: List of accounts with balances and statuses (data-testid="account-list", data-testid="account-{id}").

5. TransactionHistory.js

- *Purpose*: Shows a log of past transfers.
- *API Call*: GET /api/transactions.
- *Output*: List of transactions with date, accounts, and amount (data-testid="transaction-list", data-testid="transaction-{id}").

Styling (App.css)

- Basic centering, form layout, button styles, and list formatting.
- Color-coded success (green) and error (red) messages.

---

4. Backend Design

Directory Structure

banking-app/backend/ ├── server.js ├── package.json

API Endpoints

1. POST /api/login

- *Request*: { username: string, password: string }
- *Response*:
  - Success: 200 { success: true }
  - Failure: 401 { success: false, message: string }
- *Logic*: Checks against hardcoded users object (user: pass).

2. GET /api/accounts

- *Response*: 200 { acc123: { balance: number, status: string }, acc456: { ... } }
- *Logic*: Returns current account data.

3. GET /api/transactions

- *Response*: 200 [{ id: number, fromAccount: string, toAccount: string, amount: number, date: string }, ...]
- *Logic*: Returns transaction history array.

4. POST /api/transfer

- *Request*: { fromAccount: string, toAccount: string, amount: string }
- *Response*:
  - Success: 200 { success: true, message: string }
  - Failure: 400 { success: false, message: string }
- *Logic*: Validates transfer, updates balances, adds to transaction history.

Initial Data

- *Accounts*:
  - acc123: { balance: 1000, status: "active" }
  - acc456: { balance: 500, status: "active" }
- *Transactions*:
  - { id: 1, fromAccount: "acc123", toAccount: "acc456", amount: 200, date: "2025-03-20T10:00:00Z" }
  - { id: 2, fromAccount: "acc456", toAccount: "acc123", amount: 100, date: "2025-03-21T14:30:00Z" }
  - { id: 3, fromAccount: "acc123", toAccount: "acc456", amount: 300, date: "2025-03-22T09:15:00Z" }

---

5. Data Flow

Login

1. User enters username/password in Login.
2. POST to /api/login.
3. Backend validates credentials.
4. Frontend sets isAuthenticated and redirects to /accounts.

Account List

1. AccountList fetches /api/accounts.
2. Backend returns account data.
3. Frontend renders list with balances/statuses.

Transfer

1. User inputs fromAccount, toAccount, amount in TransferForm.
2. POST to /api/transfer.
3. Backend validates, updates accounts, adds transaction.
4. Frontend displays success/error message.

Transaction History

1. TransactionHistory fetches /api/transactions.
2. Backend returns transaction array.
3. Frontend renders list with details.

---

6. Testing Strategy

Automated Selenium Tests (generate-tests.js)

- *Purpose*: Validate end-to-end functionality (login, account list, transfer, history).
- *Techniques*:
  - *Code Extraction*: Uses @babel/parser to parse JSX from TransferForm.jsx.
  - *Embeddings*: @xenova/transformers generates vectors for caching.
  - *RAG*: Retrieves similar past code/tests for context.
  - *AI Generation*: Gemini 2.0 Flash creates test cases.
  - *Cleaning*: Removes markdown fences for executability.
- *Test Cases*:
  - Positive: Login, transfer 500, verify accounts/history.
  - Negative: Wrong login, invalid transfer (e.g., 2000).
  - Edge: Transfer 1000, zero amount.
  - History: Verify initial transactions (IDs 1-3).
- *Selectors*: data-testid attributes (e.g., fromAccount, transaction-1).
- *Execution*: Runs on http://localhost:3000/* with 5-second waits.

---

7. Deployment

Prerequisites

- Node.js, npm installed.
- Frontend: npm install in banking-app/.
- Backend: npm install in banking-app/backend/.

Running Locally

1. *Backend*: bash cd banking-app/backend npm start
2. *Frontend*: bash cd banking-app npm start
3. *Tests*: bash node generate-tests.js

9. Diagrams

System Architecture

[Browser] → [React SPA (3000)] ↑ ↓ ↑ ↓ [Selenium] [Express API (5000)] ↑ ↓ [In-Memory Store]

Data Flow (Transfer Example)

[User] → [TransferForm] → [POST /api/transfer] → [Express] ↑ ↓ ↑ ↓ ↓ [Display] ← [Response] ← [Update Accounts/Transactions]

Component Hierarchy

App ├── Login ├── AccountList ├── TransferForm └── TransactionHistory