Banking Application Design Document

1. Overview

The Banking Application is a web-based system designed to allow users to manage their bank accounts securely. It provides functionalities for user authentication (login/signup) and core banking operations (e.g., viewing balances and transferring money). This design document outlines the application's components, routes, APIs, and behaviors to support automated test generation using the `generate-tests.js` script.

- **Base URL**: `http://localhost:3000`
- **Target Audience**: End users managing personal bank accounts.
- **Key Features**: User authentication, account balance viewing, and money transfers.

---

2. Components

2.1 Login

- **Description**: The entry point for existing users to access their accounts. It's the landing page of the application.
- **Route**: `Login at /`
- **API**: `POST /api/login`
  - **Request Body**: `{ "username": string, "password": string }`
  - **Response**:
    - Success: `200 OK` with redirect to `/dashboard`
    - Failure: `401 Unauthorized` with error message
- **Behavior**:
  - Users enter their username and password in generic fields (e.g., "username field", "password field").
  - Clicking the "Login" button submits the credentials.
  - Success redirects to the Dashboard; failure displays an error message.
- **UI Elements**:
  - Username field
  - Password field
  - Login button
  - Error message display (for invalid cases)

2.2 Signup

- **Description**: Allows new users to register an account and access the application.

- **Route**: `Signup at /signup`
- **API**: `POST /api/signup`
  - **Request Body**: `{ "username": string, "password": string }`
  - **Response**:
    - Success: `201 Created` with redirect to `/dashboard`
    - Failure: `409 Conflict` (username exists) or `400 Bad Request` (invalid input)
- **Behavior**:
  - Users enter a new username and password.
  - Clicking the "Sign Up" button registers the user.
  - Success redirects to the Dashboard; failure displays an error message (e.g., "Username already exists" or "Fields cannot be empty").
- **UI Elements**: ◦ Username field ◦ Password field
  - Sign Up button
  - Error message display

2.3 Dashboard

- **Description**: The main interface for authenticated users to view their account balance and transfer money to other users.
- **Route**: `Dashboard at /dashboard`
- **API**: `POST /api/transfer`
  - **Request Body**: `{ "toUser": string, "amount": number }`
  - **Response**:
    - Success: `200 OK` with success message
    - Failure: `400 Bad Request` (invalid amount) or `402 Payment Required` (insufficient funds)
- **Requires**: Login
- **Behavior**:
  - Users must log in before accessing this page.
  - Displays the user's account balance.
  - Allows money transfers by selecting a recipient and entering an amount.
  - Success shows a confirmation message; failure shows an error (e.g., "Insufficient funds" or "Invalid amount").
- **UI Elements**:
  - Balance section (displays current balance)
  - "To User" field (dropdown or input for recipient)
  - Amount field
  - Transfer button

- Success/error message display

---

3. Application Flow

1. **Landing Page**: Users start at the Login page (`http://localhost:3000/`).
2. **Authentication**:
   - Existing users log in with valid credentials to reach the Dashboard.
   - New users navigate to Signup (`http://localhost:3000/signup`) to create an account.
3. **Post-Authentication**: Authenticated users access the Dashboard (`http://localhost:3000/dashboard`) to manage their account.

---

4. Technical Details

- **Frontend**: React.js
  - Components: `Login.js`, `Signup.js`, `Dashboard.js`
  - UI elements use generic identifiers (e.g., no `data-testid`, but `id` attributes are parsed for context).
- **Backend**: Spring Boot with MongoDB
  - APIs: `/api/login`, `/api/signup`, `/api/transfer`
- **Authentication**: Simple username/password system (no tokens for simplicity in this design).
- **Base URL**: `http://localhost:3000` (configurable via environment).
