EduFund – Full Stack Digital Banking System

By Aryabhatt Narasimha Rao Kankipati

Secure • Modular • Production-Ready

Table of Contents

1. Introduction

EduFund is a full-stack digital banking application designed to simulate real-world banking functionality. It offers role-based access for Customers and Admins, OTP-secured transactions, session handling, and a clean, component-based UI. The project reflects enterprise engineering best practices suitable for production environments.

2. Problem Statement

Most beginner banking apps are limited to CRUD operations without real security or user flow management. There's a need for a role-aware, secure, and feature-rich full-stack solution to simulate real digital banking behavior.

3. Goals & Scope

- Enable secure transactions using OTP and JWT
- Create separate flows for Admin and Customer
- Maintain proper DTO structure and controller-service separation
- Handle session expiry and account recovery
- Provide a clean, intuitive UI for all operations

4. Technology Stack

Frontend: React.js, Bootstrap, MUI

Backend: Spring Boot 3, Java 21, Spring Data JPA

Security: JWT Authentication, Spring Security

Database: H2 (Dev), MySQL (Prod)

Email/OTP: JavaMailSender, Spring @Async

Testing: Postman, Manual UI Testing

5. High-Level Architecture

The application follows a layered architecture:
- UI Components → Axios → REST API → JWT Filter → Controllers → Services → Repositories → Database
- DTOs are used throughout to map request and response bodies
- Spring Security is configured to separate public and protected routes
- Session state is managed in frontend with localStorage

6. User Journeys

👤 Customer Flow

1. Before Login:
    - / → Welcome screen
    - /customer/create → Register as a new customer
    - /customer/login → Login with email and password
    - /customer/find → Recover account via security questions
    - /transactions → Restricted (requires login)

2. After Login:
    1. JWT token and customer info stored in localStorage
    2. Access /transactions
    3. Can perform deposit/withdraw with OTP

3. OTP-Based Transaction Flow:
    1. Customer initiates a deposit or withdrawal
    2. /bank/request-otp → OTP is emailed
    3. Customer enters OTP
    4. /bank/verify-otp → If OTP is valid, transaction completes
    5. View updated transaction history securely

👨‍💼 Admin Flow

1. Before Login:
    - /admin/login → Admin login page
    - /admin, /admin/delete-account → Restricted
2. After Login:
    - adminToken stored in localStorage
    - Access /admin and /admin/delete-account
    - View all customers, accounts, and transactions
    - Delete accounts without OTP (protected via role-based JWT)

7. Core Functional Modules

- Customer Registration & Login
- Account Creation with autogenerated account number
- OTP Request & Verification
- Deposit & Withdraw APIs
- Admin Dashboard with tabbed layout
- Account Recovery using 3 security questions

8. Security & Session Management

✓ Spring Security for route protection
✓ JWT tokens with roles encoded inside
✓ `JwtAuthFilter` validates token on each request
✓ Session Timer in React auto logs out after 10 minutes
✓ Logout clears token and user data

9. API Design & Testing

- All requests are RESTful
- Token is required in Authorization header for protected routes
- OTP flow involves `/bank/request-otp` → `/bank/verify-otp`
- Postman is used to simulate all flows and validate headers, tokens, and response codes

10. Frontend Component Structure

- Home.jsx – role-aware dashboard view
- CustomerLogin.jsx – login form, token save
- Transaction.jsx – handles deposit/withdraw + OTP
- SessionTimer.jsx – auto logout logic
- AdminDashboard.jsx – tabbed admin view
- CreateCustomer.jsx – registration form with validations

11. Backend Code Architecture

- AuthController.java – Admin login flow
- CustomerController.java – register, login, get, recover
- BankingController.java – OTP, deposit, withdraw, transactions
- JwtService.java – generate and validate JWT
- SecurityConfig.java – defines which routes are public and which are protected
- OtpService.java – stores & verifies OTP against operation

12. Postman Workflow Snapshots

- Login request → receives token
- Deposit → sends token + data
- Request OTP → triggers email
- Verify OTP → returns success
- Transaction History → returns transaction list

13. Deployment Notes

- React frontend can be deployed via Netlify or Vercel
- Spring Boot backend deployable as jar or Docker container
- MySQL schema export available via Spring JPA

14. Challenges Faced

- Managing OTP state securely between requests
- Avoiding token leaks on frontend

- Ensuring session timer syncs with backend expiry
- Handling role separation in Spring Security cleanly

15. Project Impact

☑ Demonstrates full-stack capability with security, data handling, and UI/UX principles.
☑ Used by peers and juniors to understand scalable banking logic.
☑ Highlights how a real system would work with both functional and technical flows.

16. Conclusion & Learnings

This project pushed my understanding of full-stack architecture, Spring Security, OTP flows, session management, and React UI best practices. It has become a showcase project for my resume and portfolio.