📘 RETAIL BANKING WEB APPLICATION – FULL PROJECT REPORT

1. Introduction 🚀

The **Retail Banking Web Application** is a full-stack secure banking system built using:

- **Frontend:** React (Vite), Axios, React Router DOM
- **Backend:** Flask, SQLAlchemy, JWT Authentication
- **Database:** MySQL
- **Security:** OTP verification, JWT, Protected Routes
- **Tools:** Flask-Migrate, Docker (optional), Virtual Environments

This project simulates a **real-world retail banking platform** that allows customers to:

✔️ Viewing balance
✔️ Depositing and withdrawing funds
✔️ Fund transfer
✔️ Loan management
✔️ Bill payments
✔️ Transaction history
✔️ Support ticket management

This document explains the **complete project lifecycle — from idea → architecture → implementation → final deployment steps.**

---

2. Project Overview

2.1 Goal of the Project 🎯

The goal is to build a **secure, scalable, and production-ready BFSI application** that implements:

- Secure user authentication
- Accurate financial transactions
- Realistic banking workflows
- Database-driven operations
- Strong separation of frontend & backend

---

3. Features Implemented ✨

3.1 User Authentication & Security 🔐

- Registration with OTP sent through backend logic
- Email-based identity verification
- Login using JWT tokens
- Tokens stored securely (localStorage)
- Automatic redirect if token is missing or expired
- Logout + token invalidation
- Auth context in frontend for session management

3.2 Dashboard Features 🏦

- View account balance
- Deposit money
- Withdraw money
- View all transactions
- Bill payment functionality
- Loan application + admin loan approval interface
- Support ticket creation and tracking
- Transfer funds to another account

3.3 Admin Capabilities 👨‍💼

- View all loan requests
- Approve or reject loans
- View all transactions
- Manage support tickets

---

4. System Architecture 🧱

A **3-tier architecture** is used:

Frontend (React) → Backend API (Flask) → Database (MySQL)

4.1 Frontend Architecture 🎨

- React + Vite for high-speed UI development
- Axios handles API communication
- Protected routes using Auth Context
- Components separated into:

```
pages/
components/
axios.js
assets/
```

UI Pages include:

- Login
- Register
- Verify OTP
- Dashboard
- Deposit / Withdraw
- Fund Transfer
- Transactions
- Loans
- Bill Payments
- Support

---

5. Backend Architecture ⚙️

The backend follows modular Flask Blueprint structure:

```
backend/
│── app/
│   ├── models/
│   ├── routes/
│   └── __init__.py
│── migrations/
│── requirements.txt
│── run.py
```

5.1 Backend Modules

5.1.1 Models 📄

- **User** – stores user details, password hash
- **Account** – stores account balance
- **Transaction** – deposit, withdrawal, transfer
- **Loan** – loan request and approval
- **Bill** – bill payment history
- **Support** – customer support tickets

5.1.2 Routes (APIs) 🔗

- `/auth` – register, OTP, login
- `/account` – balance, deposit, withdraw
- `/transaction` – all transactions, fund transfer
- `/loan` – apply, admin approve
- `/bill` – pay bills
- `/support` – raise support requests

---

6. Database Design 🗄️

6.1 Tables

Table	Description
user	Stores details for authentication
account	Stores balance using SQLAlchemy Numeric()
transaction	Logs all financial transactions
otp	Temporary OTPs for registration
loan	Loan applications and statuses
bill	Bill payment records
support	User-raised support tickets

6.2 Data Types Used

- **Numeric(10, 2)** for balance → avoids float errors
- **DateTime** for timestamps

- **Strings** for user details
- **Foreign keys** for account → user mapping

---

7. Security Features 🔒

7.1 OTP-Based Registration

User registers → backend generates OTP → user verifies → account created.

7.2 JWT Authentication

- Tokens issued during login
- Stored in localStorage
- Validated before loading dashboard

7.3 Protected Routes

Only authenticated users can access dashboard features.

7.4 Logout

Clears token + invalidates session.

---

8. Development Workflow 🛠️

8.1 Virtual Environment Setup

Backend uses **clean venv** (not committed to Git).

8.2 Requirements Installation

```
pip install -r requirements.txt
```

8.3 Frontend Installation

```
npm install
npm run dev
```

8.4 Database Migration

```
flask db init
flask db migrate -m "Initial"
flask db upgrade
```

---

9. Running the Application ▶️

The project includes **scripts** to simplify development.

9.1 Backend Setup

Step 1: Create Virtual Environment

```
python3 -m venv venv
source venv/bin/activate
```

Step 2: Install Dependencies

```
pip install -r requirements.txt
```

Step 3: Configure Environment Variables

Edit:

```
backend/.env
```

Set:

```
DATABASE_URL=mysql+pymysql://root:password@localhost/banking_app
SECRET_KEY=your_secret_key_here
FLASK_ENV=development
FLASK_APP=run.py
```

Step 4: Apply Migrations

```
flask db upgrade
```

Step 5: Start Backend

```
python run.py
```

Backend runs at:
👉 http://127.0.0.1:5000

9.2 Frontend Setup

Inside `frontend/`:

Step 1: Install Node Modules

```
npm install
```

Step 2: Run Development Server

```
npm run dev
```

Frontend runs at:
👉 http://localhost:5173

---

10. Scripts Used in the Project 📜

Stored inside `scripts/` folder.

10.1 setup.sh

```
#!/bin/bash
cd backend

python3 -m venv venv
source venv/bin/activate

pip install -r requirements.txt

echo "Backend environment setup complete."
```

10.2 run-dev.sh

```
#!/bin/bash

echo "Starting Backend..."
cd backend
source venv/bin/activate
python run.py &

echo "Starting Frontend..."
cd ../frontend
npm run dev
```

---

11. Testing Workflow 🧪

11.1 Backend Tests (Postman)

Feature	Test Details
Registration	OTP generation, validation
Login	JWT, incorrect password handling
Balance	Numeric precision verified
Deposit	Works correctly

Feature	Test Details
Withdraw	Cannot go negative
Fund Transfer	Sender/receiver updates
Loan	Apply + approve
Support	Ticket creation

All endpoints returned **correct JSON**.

11.2 Frontend Tests

Area	Result
Auth Redirect	Works
Token Expiry	Forced logout
Transaction Inputs	Validation correct
Dark Mode	Working
Navigation	Protected routes working

---

12. Deployment Readiness 🚀

This project can be deployed on:

- AWS EC2
- Azure VM
- DigitalOcean
- Railways
- Render
- Docker

12.1 Production Build Steps

Frontend Build

```
npm run build
```

Backend Production

```
gunicorn run:app
```

(Docker containerization optional)

---

13. Repository Structure Overview 📂

Final clean repo includes:

```
Retail_Banking_App/
├── backend/
├── frontend/
├── docs/
├── scripts/
├── tests/
├── README.md
└── LICENSE
```

Removed:
❌ venv/
❌ node_modules/
❌ **pycache**/

---

14. Challenges Faced & Solutions 💡

14.1 Float Inaccuracy

✔️ Solved by Numeric(10,2)

14.2 JWT Not Clearing

✔️ Cleared token on session load

14.3 Balance Not Showing

✔️ Fixed props + fetch logic

14.4 Support Form Not Working

✔️ Corrected backend + axios

---

15. Conclusion 🎉

The **Retail Banking Web Application** is a:

🌟 Complete
🌟 Secure
🌟 Production-ready
🌟 BFSI-grade full-stack application