🏦 Co-operative Society Bank – Software Project Blueprint

1. Project Overview

A digital banking platform for a Co-operative Society that manages members, deposits, loans, accounts, and reporting with web and mobile access.

2. Core Modules

◇ Member Management

- Member registration with KYC (Aadhar, PAN, etc.)
- Membership fee collection
- Member profile management

◇ Accounts & Deposits

- Savings Account (Interest calculation, passbook)
- Fixed Deposit / Recurring Deposit schemes
- Withdrawal and deposit management

◇ Loan Management

- Loan application & approval workflow
- EMI calculation & repayment tracking
- Interest calculation & penalty management

◇ Transactions

- Cash deposit/withdrawal
- Fund transfer (internal + external via UPI/NEFT/IMPS)

- Transaction history

◇ Reports & Dashboard

- Daily/Monthly financial reports
- Loan outstanding reports
- Profit/Loss and member dividend reports

◇ Admin & Compliance

- Role-based access (Admin, Staff, Auditor)
- Audit logs of transactions
- RBI/Co-operative compliance reports

3. Technology Stack

- **Frontend**: React.js + Redux + Material UI
- **Backend**: Node.js + Express.js
- **Database**: MongoDB / PostgreSQL (depends on requirement)
- **Authentication**: JWT + Secure Cookies + Role-based auth
- **Payments/Banking APIs**: NPCI UPI API, Razorpay/Paytm integration for digital transactions
- **Deployment**: Docker + AWS/GCP (scalable infra)

4. System Architecture

Client (React App) ⟶ REST API (Node.js/Express) ⟶ Database (MongoDB/PostgreSQL)

- Authentication Layer (JWT + OAuth if needed)
- File Storage (AWS S3 / Cloudinary for documents)
- Notification Service (Email, SMS, WhatsApp)

5. Security & Compliance

- End-to-End encryption (HTTPS/TLS)
- Secure password storage (bcrypt)
- Two-factor authentication (OTP via SMS/Email)
- RBI & Co-operative society compliance

6. Roadmap (Phases)

1. **MVP –** Member mgmt, basic savings, manual loan tracking
2. **Phase 2 –** Loan automation, EMI calculator, dashboards
3. **Phase 3 –** Digital payments (UPI/NEFT/IMPS integration)
4. **Phase 4 –** Mobile app, AI-based credit risk scoring

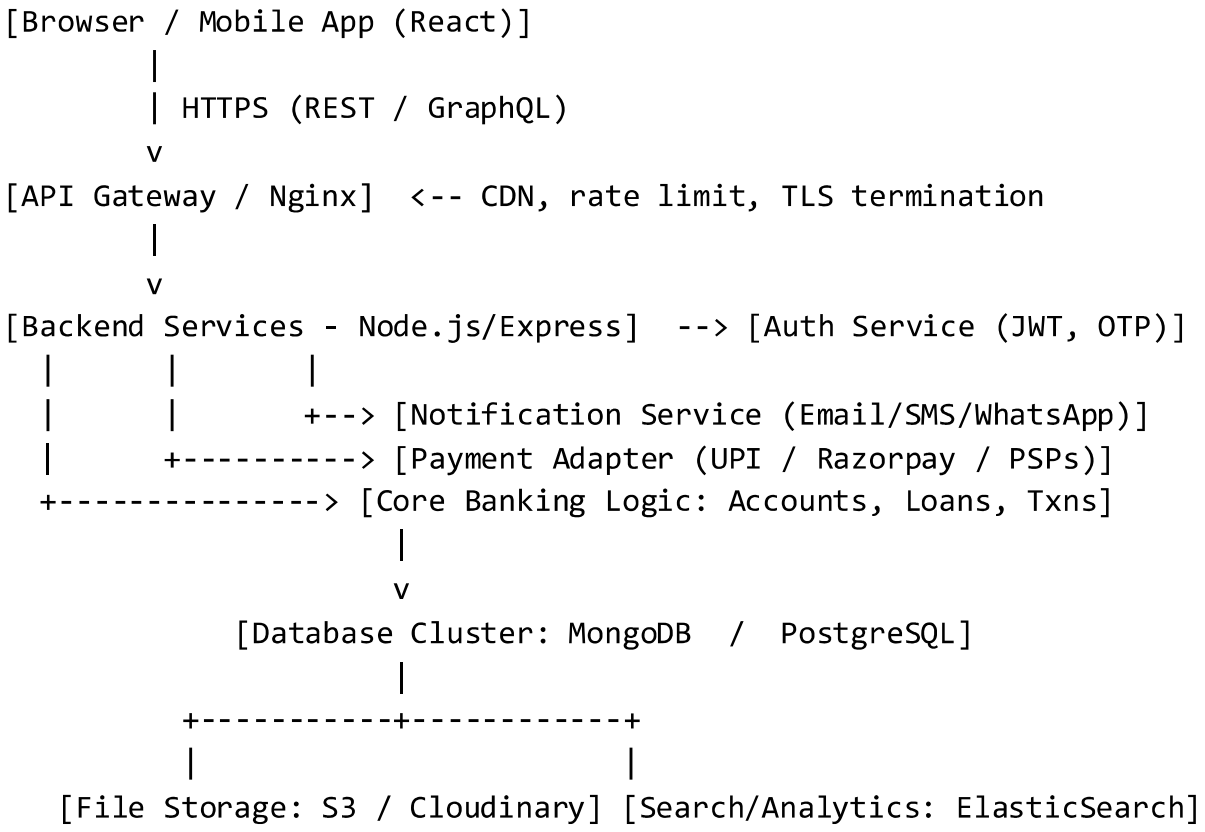
7. Deliverables for Client

- Web Application (Admin + Member Portal)
- Mobile Responsive UI (React)
- API Documentation (Swagger/Postman)
- Deployment & Training

Architecture Design — Co-operative Society Bank (React + Node.js)

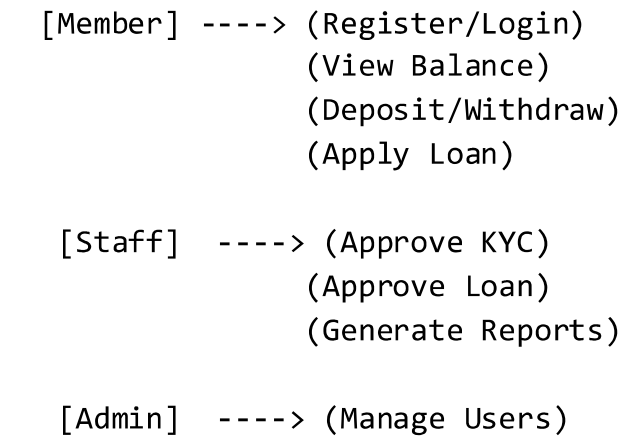
Nice — below is a complete, practical architecture design you can use in your meeting. It includes a high-level architecture diagram (ASCII), component breakdown, data models (Mongo + SQL variants), key API endpoints (examples), authentication & security, deployment & scaling, monitoring, CI/CD, and recommended folder structure. Use what you need — I kept it actionable and developer-friendly.

High-level architecture (text diagram)



◇ 1. Use Case Diagram (high-level features)

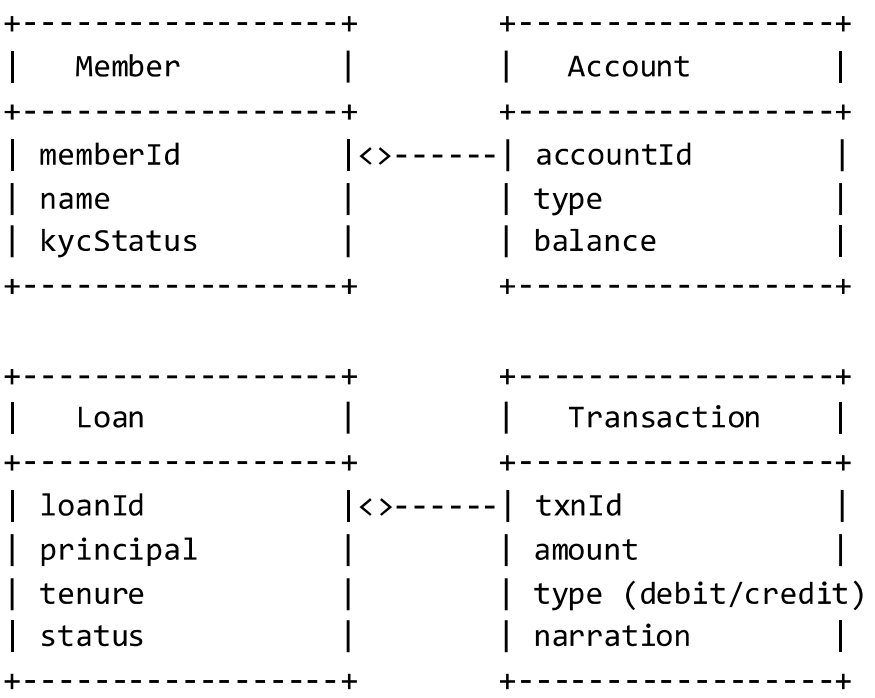
Shows actors (Member, Staff, Admin) and what they can do.



```
(System Settings)
(Audit Logs)
```

◇ 2. Class Diagram (core entities)

Represents main backend models.



◇ 3. Sequence Diagram (Loan Application Flow)

Shows interactions step by step.

```
Member -> Frontend : Apply Loan
Frontend -> Backend API : POST /loans
Backend -> DB : Save Loan (status=pending)
Backend -> Staff Portal : Notify for approval
Staff -> Backend API : Approve Loan
Backend -> DB : Update Loan (status=approved)
```

```
Backend -> Notification Service : Send SMS/Email
```

◇ 4. Deployment Diagram

How the system will be deployed.

```
[Client Browser]
       |
 [Load Balancer / API Gateway]
       |
+-----------------------------+
| Backend (Node.js, Express)  |
|  - Auth Service             |
|  - Loan Service             |
|  - Account Service          |
+-----------------------------+
       |
+-----------+   +-------------------+
| Database  |   | Storage (S3, etc.)|
| Postgres  |   | Documents, Images |
+-----------+   +-------------------+
```

Components & Responsibilities

1. **Frontend (React)**
    a. Admin portal (staff): account management, KYC review, approvals, reports
    b. Member portal (responsive): registration, balance, statements, loan apply, repayments
    c. State management: Redux Toolkit / React Query (for caching)
    d. UI library: Material UI (you already use it)
2. **API Layer (Node.js + Express / Fastify)**
    a. RESTful endpoints (or GraphQL if you prefer single endpoint)
    b. Input validation (Joi / Zod)
    c. Rate limiting, CORS, request logging (morgan/winston/pino)

    d. Error handling, standardized API responses
3. **Auth Service**
    a. JWT for session tokens (access + refresh)
    b. OTP service for sensitive operations (login, transaction)
    c. Role-based access control (RBAC): roles = [member, staff, manager, auditor, admin]
4. **Core Banking Module**
    a. Accounts service: create accounts, calculate interest, passbook
    b. Loan service: application workflow, credit checks, amortization, EMI schedule
    c. Transactions service: ledger entries, reconciliation, idempotency
5. **Payment & Integration Adapter**
    a. UPI / PSP integration via gateway (NPCI via bank/aggregator) and Razorpay/Paytm for wallets/cards
    b. Webhooks for asynchronous events
6. **Notification Service**
    a. Email (SES/SendGrid), SMS gateway, WhatsApp (third-party)
    b. Transactional templates and alerts
7. **Storage & DB**
    a. Document store for KYC docs: S3/Cloudinary
    b. Primary DB: MongoDB (schemaless, quicker iteration) OR PostgreSQL (ACID, stronger relational integrity)
    c. Secondary: ElasticSearch for fast search & analytics
8. **Monitoring & Security**
    a. Prometheus + Grafana for metrics
    b. ELK stack / Loki for logs
    c. WAF, TLS, automated backups, audit logs

Data models

Below are suggested models for MongoDB (JSON-like). If you use PostgreSQL, convert types to columns and add proper relations and constraints.

Member (members)

```
{
  "_id": "ObjectId",
  "memberId": "MSR-0001",
  "name": "Satyam Ray",
  "email": "user@example.com",
  "mobile": "+919XXXXXXXXX",
  "dob": "1988-01-01",
  "address": {...},
  "kyc": {
    "aadhar": "xxxx-xxxx-xxxx",
    "pan": "ABCDE1234F",
    "documents": [
      {"type":"aadhar","url":"s3://...","status":"verified"}
    ],
    "verified": false
  },
  "joinedAt": "2025-09-16T12:00:00Z",
  "role": "member",
  "status": "active"
}
```

Account (accounts)

```
{
  "_id": "ObjectId",
  "accountId": "ACC-10001",
  "memberId": "MSR-0001",
  "type": "savings", // savings, fd, rd, current
  "balance": 12500.50,
  "interestRate": 4.0,
  "createdAt": "...",
  "status": "active",
  "metadata": {...}
}
```

Loan (loans)

```
{
  "_id":"ObjectId",
  "loanId":"LN-20001",
  "memberId":"MSR-0001",
  "product":"personal", // agriculture, housing, business, etc.
  "principal":500000,
  "interestRate":10.5,
  "tenureMonths":60,
  "emi":10624.50,
  "amortizationSchedule":[
     {"dueDate":"2025-10-
01","principal":...,"interest":...,"status":"pending"}
  ],
  "status":"approved", // applied, approved, disbursed, closed
  "createdAt":"..."
}
```

Transaction / Ledger (transactions)

```
{
  "_id":"ObjectId",
  "txnId":"TXN-900001",
  "accountId":"ACC-10001",
  "memberId":"MSR-0001",
  "type":"credit", // debit
  "mode":"cash|upi|neft|cheque|internal",
  "amount":5000,
  "balanceAfter":17500.50,
  "narration":"Deposit",
  "createdAt":"..."
}
```

Audit log (audit_logs)

```
{
  "_id":"ObjectId",
  "entity":"transactions",
  "entityId":"TXN-900001",
  "action":"create",
  "performedBy":"staff-1",
  "timestamp":"...",
  "diff": {...}
}
```

Database design notes

- If using **PostgreSQL**, enforce ACID for transactional integrity:
  - tables: members, accounts, loans, transactions, audit_logs, kyc_documents
  - use SERIAL/UUID for ids, foreign keys for member->accounts->transactions
  - use DB transactions when creating loan + disbursement + ledger entries
- If using **MongoDB**:
  - use transactions for multi-document updates (replica set)
  - keep immutable ledger entries (append-only) for transactions
  - index on memberId, accountId, txnId, loanId for performance

Key API endpoints (REST examples)

Authentication:

- `POST /api/v1/auth/register` — register member (returns pending KYC)
- `POST /api/v1/auth/login` — login (email/mobile + password) -> returns access + refresh tokens
- `POST /api/v1/auth/otp` — send/verify OTP

Members:

- `GET /api/v1/members/:id`
- `POST /api/v1/members/:id/kyc` — upload documents

Accounts:

- `POST /api/v1/accounts` — create account (staff or automatic)
- `GET /api/v1/accounts/:accountId`
- `POST /api/v1/accounts/:accountId/deposit`
- `POST /api/v1/accounts/:accountId/withdraw`

Loans:

- `POST /api/v1/loans` — apply for loan
- `GET /api/v1/loans/:loanId`
- `POST /api/v1/loans/:loanId/approve` — staff action
- `POST /api/v1/loans/:loanId/disburse`

Transactions:

- `GET /api/v1/transactions?memberId=...&from=...&to=...`
- Webhook: `POST /api/v1/payments/webhook` — payment provider callbacks

Admin:

- `GET /api/v1/reports/daily` — protected, role=manager/auditor

Example: Loan EMI calculation (pseudo)

Monthly EMI formula:

```
r = annualRate / 12 / 100
n = tenureMonths
EMI = P * r * (1+r)^n / ((1+r)^n - 1)
```

Implement this same calculation in backend to generate amortization schedule and store each installment as a scheduled ledger entry.

Auth & Security

- Use HTTPS everywhere (TLS 1.2+)
- JWT best-practice: short-lived access token (e.g., 15 min), long-lived refresh token stored in secure HTTP-only cookie.
- Protect refresh token endpoints with CSRF tokens.
- Password hashing: bcrypt (work factor configurable), or Argon2 if available.
- MFA: OTP via SMS/email on sensitive ops (login from new device, large transfer).
- Audit logs for any financial operations (who, when, why).
- Rate limiting per IP and per account for critical endpoints (login, transfer).
- Input validation and sanitize user input to avoid injection.
- Regular backups and encryption at rest for DB and S3.

Reliability & Scaling

- Deploy backend as stateless containers behind a load balancer (AWS ALB / GCP LB).
- Use managed DB (Amazon RDS for PostgreSQL / MongoDB Atlas) with replicas.
- Use S3 for file storage, enable lifecycle & versioning.
- Use Redis for session/cache and distributed locks (e.g., for duplicate payment prevention).
- Use job queue (BullMQ / RabbitMQ) for background tasks (send email/sms, run interest accrual jobs).
- For HATM-like operations (interest accrual), run scheduled jobs (cron) in dedicated worker.

Folder structure (recommended)

Backend (Node.js):

```
/backend
  /src
    /controllers
    /services
    /routes
    /models
    /jobs
    /utils
    /middlewares
  /tests
  Dockerfile
  package.json
```

Frontend (React):

```
/frontend
  /src
    /components
    /pages
    /hooks
    /services  // API calls
    /store    // Redux or Zustand
    /utils
  public/
  package.json
  Dockerfile
```

UX / Data flow suggestions for meeting

- On registration: member submits basic details -> temporary account -> upload KYC docs -> staff reviews & verifies -> account activated.

- Loan flow: apply (frontend) -> basic eligibility check (score) -> staff verification -> approval -> disbursement -> ledger entries & notification.
- Show live demo of "Apply Loan" screen and "Approve Loan" in admin to demonstrate flows.

Bonus — Quick System Architecture Diagram (Mermaid-style, for docs)

```
A[React Frontend] -->|HTTPS| B[API Gateway / Nginx]
B --> C[Node.js / Express API]
C --> D[(Database: MongoDB/Postgres)]
C --> E[S3 / Cloud Storage]
C --> F[Payment Adapter (UPI/Razorpay)]
C --> G[Notification Service]
C --> H[ElasticSearch]
```