DiGiBachat - Group Savings & Lending Platform

Complete Project Plan & Technical Blueprint

🎯 Project Overview

DiGiBachat is a fintech platform enabling group-based savings and lending with secure mobile-first authentication, UPI integration, and comprehensive financial management tools.

Vision: Democratize group savings and lending through technology, making it accessible, transparent, and secure for communities.

---

1. 🔐 Feature Breakdown by User Roles

Admin Role

- **Group Management**
  - Create new savings groups with customizable rules
  - Set group contribution amounts and frequencies
  - Define loan interest rates and repayment terms
  - Archive or dissolve groups when needed
  - Export group data and reports
- **Member Administration**
  - Approve/deny membership requests
  - Remove members from groups
  - Assign Treasurer role to trusted members
  - View complete member activity logs
  - Send bulk notifications to group members
- **Financial Controls**
  - Set minimum/maximum loan amounts
  - Configure autopay settings for the group

    - Approve loan applications above threshold amounts
    - Generate and download financial reports (PDF)
    - Monitor group's overall financial health

Treasurer Role

- **Day-to-Day Management**
    - Approve/deny loan applications within limits
    - Track member contributions and dues
    - Generate monthly/weekly invoices
    - Send payment reminders to members
    - Maintain transaction records
- **Financial Reporting**
    - Create periodic financial statements
    - Track interest earnings and distributions
    - Monitor overdue payments and defaults
    - Generate member-wise contribution reports
    - Export transaction history

Member Role

- **Personal Finance**
    - View personal savings balance and history
    - Apply for loans from group pool
    - Make contributions via UPI/Autopay
    - Track loan repayments and interest
    - Download personal financial statements
- **Group Interaction**
    - Join groups using invite codes
    - View group's total savings (without individual details)
    - Participate in group decisions (via voting features)
    - Receive notifications for dues and updates
    - Access group policies and rules

---

2. 🛠️ Recommended Tech Stack

Frontend Technologies

```
Web Application:
├── React 18+ (with TypeScript)
├── TailwindCSS + Headless UI
├── React Router v6 for navigation
├── React Query for state management
├── React Hook Form for forms
├── Chart.js/Recharts for analytics
├── jsPDF for report generation
└── Axios for API calls

Mobile Application:
├── React Native 0.72+
├── Expo SDK 49+
├── React Navigation v6
├── React Native Paper (Material Design)
├── AsyncStorage for local data
├── React Native UPI for payments
├── Push Notifications (Expo)
└── React Native PDF for reports
```

Backend Technologies

```
Server:
├── Node.js 18+ with Express.js
├── TypeScript for type safety
├── Helmet.js for security headers
├── Express Rate Limit for API protection
├── Morgan for logging
├── Joi/Zod for validation
├── JWT for authentication
└── bcrypt for password hashing

Database & Storage:
├── PostgreSQL 15+ (NeonDB)
├── Prisma ORM for database operations
├── Redis for caching and sessions
├── AWS S3 for file storage
└── Cloudinary for image processing
```

Third-Party Integrations

```
Authentication:
├── Twilio Verify API for OTP
```

```
└── Firebase Auth (backup option)

Payments:
├── Razorpay API (Primary)
├── Cashfree API (Secondary)
└── UPI Deep Linking

Notifications:
├── Twilio for SMS
├── Firebase Cloud Messaging
├── SendGrid for emails
└── WhatsApp Business API

Monitoring:
├── Sentry for error tracking
├── New Relic for performance
└── LogRocket for user sessions
```

Deployment & Infrastructure

```
Backend Deployment:
├── AWS EC2 (Production)
├── Railway/Render (Development)
├── Docker containers
└── PM2 for process management

Frontend Deployment:
├── Vercel (Web app)
├── Expo EAS Build (Mobile)
└── CDN via Cloudflare

Database:
├── NeonDB (Serverless PostgreSQL)
├── Redis Cloud for caching
└── Automated backups
```

---

3. 📁 Project Structure (Monorepo Approach)

```
digibachat/
├── packages/
│   ├── web-app/              # React web application
│   │   ├── public/
│   │   ├── src/
│   │   │   ├── components/
│   │   │   │   ├── ui/           # Reusable UI components
│   │   │   │   ├── forms/        # Form components
│   │   │   │   ├── charts/       # Analytics components
│   │   │   │   └── layout/       # Layout components
│   │   │   ├── pages/
│   │   │   │   ├── auth/         # Login/Signup pages
│   │   │   │   ├── dashboard/    # Dashboard pages
│   │   │   │   ├── groups/       # Group management
│   │   │   │   ├── transactions/ # Financial pages
│   │   │   │   └── reports/      # Reports and analytics
│   │   │   ├── hooks/            # Custom React hooks
│   │   │   ├── services/         # API service functions
│   │   │   ├── store/            # Global state management
│   │   │   ├── utils/            # Utility functions
│   │   │   └── types/            # TypeScript definitions
│   │   ├── tailwind.config.js
│   │   └── package.json
│   │
│   ├── mobile-app/               # React Native application
│   │   ├── src/
│   │   │   ├── components/
│   │   │   │   ├── ui/           # Mobile UI components
│   │   │   │   ├── forms/        # Mobile forms
│   │   │   │   └── navigation/   # Navigation components
│   │   │   ├── screens/
│   │   │   │   ├── Auth/         # Authentication screens
│   │   │   │   ├── Dashboard/    # Dashboard screens
│   │   │   │   ├── Groups/       # Group screens
│   │   │   │   ├── Transactions/ # Transaction screens
│   │   │   │   └── Profile/      # Profile screens
│   │   │   ├── services/         # API and native services
│   │   │   ├── store/            # Redux/Zustand store
│   │   │   ├── utils/            # Mobile utilities
│   │   │   └── types/            # Shared types
│   │   ├── app.json
│   │   └── package.json
│   │
│   ├── backend/                  # Node.js backend
│   │   ├── src/
│   │   │   ├── controllers/
│   │   │   │   ├── auth.controller.ts
│   │   │   │   ├── group.controller.ts
│   │   │   │   ├── transaction.controller.ts
│   │   │   │   ├── loan.controller.ts
│   │   │   │   └── report.controller.ts
│   │   │   ├── middleware/
│   │   │   │   ├── auth.middleware.ts
│   │   │   │   ├── validation.middleware.ts
│   │   │   │   ├── rateLimiting.middleware.ts
│   │   │   │   └── errorHandler.middleware.ts
│   │   │   ├── models/          # Prisma models
│   │   │   ├── routes/
│   │   │   │   ├── auth.routes.ts
│   │   │   │   ├── group.routes.ts
│   │   │   │   ├── transaction.routes.ts
│   │   │   │   └── report.routes.ts
│   │   │   ├── services/
│   │   │   │   ├── auth.service.ts
│   │   │   │   ├── payment.service.ts
│   │   │   │   ├── notification.service.ts
│   │   │   │   └── report.service.ts
│   │   │   ├── utils/
│   │   │   │   ├── encryption.ts
│   │   │   │   ├── validation.ts
│   │   │   │   └── logger.ts
│   │   │   └── types/          # Backend types
│   │   ├── prisma/
│   │   │   ├── schema.prisma
│   │   │   └── migrations/
│   │   ├── tests/
│   │   └── package.json
│   │
│   └── shared/                 # Shared utilities and types
│       ├── types/
│       │   ├── user.types.ts
│       │   ├── group.types.ts
│       │   ├── transaction.types.ts
│       │   └── api.types.ts
│       ├── utils/
│       │   ├── constants.ts
│       │   ├── validators.ts
│       │   └── formatters.ts
│       └── package.json
│
├── tools/                      # Build and development tools
├── docs/                       # Project documentation
├── package.json              # Root package.json
├── turbo.json               # Turborepo configuration
└── README.md
```

---

4. 🔌 API Design & Endpoints

Authentication APIs

```
POST /api/auth/send-otp
POST /api/auth/verify-otp
POST /api/auth/refresh-token
POST /api/auth/logout
GET  /api/auth/profile
PUT  /api/auth/profile
```

Group Management APIs

```
POST /api/groups                 # Create new group
GET  /api/groups                 # Get user's groups
GET  /api/groups/:id             # Get group details
PUT  /api/groups/:id             # Update group settings
DELETE /api/groups/:id           # Delete group

POST /api/groups/:id/join         # Join group with code
POST /api/groups/:id/invite       # Generate invite code
POST /api/groups/:id/approve/:userId # Approve member request
DELETE /api/groups/:id/members/:userId # Remove member
PUT  /api/groups/:id/members/:userId/role # Update member role
```

Savings & Transaction APIs

```
POST /api/transactions/contribute   # Make contribution
GET  /api/transactions/history      # Get transaction history
GET  /api/transactions/balance      # Get current balance
POST /api/transactions/autopay      # Setup autopay

GET  /api/groups/:id/transactions   # Group transaction history
GET  /api/groups/:id/balance        # Group total balance
```

Loan Management APIs

```
POST /api/loans/apply              # Apply for loan
GET  /api/loans/applications        # Get loan applications
PUT  /api/loans/:id/approve         # Approve loan
POST /api/loans/:id/repay          # Make loan repayment
GET  /api/loans/:id/schedule       # Get repayment schedule
```

Reports & Analytics APIs

```
GET  /api/reports/group/:id        # Group financial report
GET  /api/reports/user             # User financial report
POST /api/reports/generate          # Generate PDF report
GET  /api/analytics/dashboard       # Dashboard analytics
```

Notification APIs

```
POST /api/notifications/send        # Send notification
GET  /api/notifications             # Get user notifications
PUT  /api/notifications/:id/read    # Mark as read
PUT  /api/notifications/settings    # Update notification preferences
```

---

5. 🗃️ Database Schema (PostgreSQL)

```
-- Users table
CREATE TABLE users (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    phone_number VARCHAR(15) UNIQUE NOT NULL,
    name VARCHAR(100) NOT NULL,
    email VARCHAR(255),
    profile_picture_url TEXT,
    is_verified BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Groups table
CREATE TABLE groups (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    name VARCHAR(100) NOT NULL,
    description TEXT,
    group_code VARCHAR(8) UNIQUE NOT NULL,
    contribution_amount DECIMAL(10,2) NOT NULL,
    contribution_frequency VARCHAR(20) NOT NULL, -- weekly/monthly
    max_members INTEGER DEFAULT 50,
    loan_interest_rate DECIMAL(5,2) DEFAULT 2.0,
    created_by UUID REFERENCES users(id),
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Group members with roles
CREATE TABLE group_members (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    group_id UUID REFERENCES groups(id) ON DELETE CASCADE,
    user_id UUID REFERENCES users(id) ON DELETE CASCADE,
    role VARCHAR(20) DEFAULT 'member', -- admin/treasurer/member
    status VARCHAR(20) DEFAULT 'pending', -- pending/active/inactive
    joined_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(group_id, user_id)
);

-- Savings transactions
CREATE TABLE savings_transactions (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    group_id UUID REFERENCES groups(id),
    user_id UUID REFERENCES users(id),
    amount DECIMAL(10,2) NOT NULL,
    transaction_type VARCHAR(20) NOT NULL, -- contribution/withdrawal
    payment_method VARCHAR(50), -- upi/autopay/manual
    payment_reference VARCHAR(100),
    status VARCHAR(20) DEFAULT 'pending', -- pending/completed/failed
    transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    description TEXT
);

-- Loan transactions
CREATE TABLE loans (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    group_id UUID REFERENCES groups(id),
    borrower_id UUID REFERENCES users(id),
    amount DECIMAL(10,2) NOT NULL,
    interest_rate DECIMAL(5,2) NOT NULL,
    tenure_months INTEGER NOT NULL,
    monthly_emi DECIMAL(10,2) NOT NULL,
    status VARCHAR(20) DEFAULT 'pending', -- pending/approved/active/completed/defaulted
    applied_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    approved_at TIMESTAMP,
    approved_by UUID REFERENCES users(id),
    disbursed_at TIMESTAMP
);

-- Loan repayments
CREATE TABLE loan_repayments (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    loan_id UUID REFERENCES loans(id),
    amount DECIMAL(10,2) NOT NULL,
    principal_amount DECIMAL(10,2) NOT NULL,
    interest_amount DECIMAL(10,2) NOT NULL,
    payment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    payment_method VARCHAR(50),
    payment_reference VARCHAR(100),
    status VARCHAR(20) DEFAULT 'completed'
);

-- Financial reports
CREATE TABLE reports (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    group_id UUID REFERENCES groups(id),
    generated_by UUID REFERENCES users(id),
    report_type VARCHAR(50) NOT NULL, -- monthly/quarterly/yearly
    report_period_start DATE NOT NULL,
    report_period_end DATE NOT NULL,
    file_url TEXT,
    generated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Notifications
CREATE TABLE notifications (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id),
    group_id UUID REFERENCES groups(id),
    title VARCHAR(200) NOT NULL,
    message TEXT NOT NULL,
    type VARCHAR(50) NOT NULL, -- reminder/alert/update
    is_read BOOLEAN DEFAULT FALSE,
    sent_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Audit logs for compliance
```

```
CREATE TABLE audit_logs (
    id UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    user_id UUID REFERENCES users(id),
    action VARCHAR(100) NOT NULL,
    resource_type VARCHAR(50) NOT NULL,
    resource_id UUID,
    details JSONB,
    ip_address INET,
    user_agent TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Indexes for performance
CREATE INDEX idx_group_members_group_id ON group_members(group_id);
CREATE INDEX idx_group_members_user_id ON group_members(user_id);
CREATE INDEX idx_savings_transactions_group_id ON savings_transactions(group_id);
CREATE INDEX idx_savings_transactions_user_id ON savings_transactions(user_id);
CREATE INDEX idx_loans_group_id ON loans(group_id);
CREATE INDEX idx_loans_borrower_id ON loans(borrower_id);
CREATE INDEX idx_notifications_user_id ON notifications(user_id);
CREATE INDEX idx_audit_logs_user_id ON audit_logs(user_id);
```

---

6. 🔒 Security & Compliance

Authentication & Authorization

```
// JWT Token Structure
interface JWTPayload {
  userId: string;
  phoneNumber: string;
  roles: string[];
  iat: number;
  exp: number;
}

// Role-based access control
const permissions = {
  admin: ['*'], // Full access
  treasurer: ['approve_loans', 'generate_reports', 'manage_members'],
  member: ['view_own_data', 'apply_loan', 'make_contribution']
};
```

Data Encryption

- **In Transit**: TLS 1.3 for all API communications
- **At Rest**: PostgreSQL column-level encryption for sensitive data
- **PII Protection**: Phone numbers and financial data encrypted
- **Key Management**: AWS KMS for encryption key rotation

Security Headers & Middleware

```
// Express security configuration
app.use(helmet({
  contentSecurityPolicy: {
    directives: {
      defaultSrc: ["'self'"],
      scriptSrc: ["'self'", "'unsafe-inline'"],
      styleSrc: ["'self'", "'unsafe-inline'"],
      imgSrc: ["'self'", "data:", "https:"]
    }
  }
}));

// Rate limiting
app.use('/api/', rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
  max: 100 // limit each IP to 100 requests per windowMs
}));
```

Compliance & Audit

- **Data Retention**: 7-year transaction history retention
- **Audit Logging**: All financial transactions logged
- **KYC Integration**: Ready for regulatory compliance
- **Backup Strategy**: Daily automated backups with 30-day retention
- **GDPR Compliance**: Data anonymization and deletion capabilities

---

7. 🚀 Development Roadmap

Phase 1: MVP

Goal: Launch basic group savings functionality

Features:

- ✅ User authentication via OTP
- ✅ Group creation and joining with codes
- ✅ Basic member management
- ✅ Savings contributions tracking
- ✅ Simple UPI payment integration
- ✅ Basic transaction history
- ✅ Mobile-responsive web app

Deliverables:

- Web application (React)
- Backend API (Node.js + PostgreSQL)
- Basic mobile app (React Native)
- Payment gateway integration
- User testing with 3-5 pilot groups

Phase 2: Core Features

Goal: Complete core lending and reporting features

Features:

- ✅ Loan application and approval system
- ✅ EMI calculation and repayment tracking
- ✅ Invoice generation and PDF reports
- ✅ SMS/Email notifications
- ✅ Treasurer role and permissions
- ✅ Advanced transaction filtering
- ✅ Group analytics dashboard

Deliverables:

- Full-featured mobile app
- Automated notification system
- Comprehensive reporting module
- Admin panel for group management
- Beta testing with 25+ groups

Phase 3: Advanced Features

Goal: Scale and optimize for growth

Features:

- ✅ Autopay integration for recurring contributions
- ✅ Advanced analytics and insights
- ✅ Multi-group membership for users
- ✅ Bulk operations for treasurers
- ✅ API rate limiting and caching
- ✅ Performance optimization
- ✅ Security hardening

Deliverables:

- Production-ready platform
- Performance benchmarks (1000+ concurrent users)
- Security audit completion
- App Store/Play Store launch
- Customer support system

Phase 4: Scale & Innovation

Goal: Market expansion and AI integration

Features:

- 🔮 AI-powered credit scoring
- 🔮 Fraud detection algorithms
- 🔮 Gamification features
- 🔮 Investment options for group savings
- 🔮 Regulatory compliance automation
- 🔮 Multi-language support
- 🔮 WhatsApp bot integration

Deliverables:

- Enterprise-grade platform
- Regulatory compliance certification
- Multi-region deployment
- Partnership integrations
- Advanced AI features

---

8. 💡 Recommendations for Robustness

Performance Optimization

// Database query optimization

```
const optimizedGroupQuery = await prisma.group.findMany({
  where: { id: groupId },
  include: {
    members: {
      select: { id: true, name: true, role: true, status: true }
    },
    _count: {
      select: {
        savings_transactions: true,
        loans: true
      }
    }
  }
});

// Caching strategy with Redis
const cacheKey = `group_${groupId}_summary`;
let groupSummary = await redis.get(cacheKey);
if (!groupSummary) {
  groupSummary = await calculateGroupSummary(groupId);
  await redis.setex(cacheKey, 300, JSON.stringify(groupSummary)); // 5-min cache
}
```

Error Handling & Monitoring

```
// Global error handler
app.use((error: Error, req: Request, res: Response, next: NextFunction) => {
  logger.error('Unhandled error:', {
    error: error.message,
    stack: error.stack,
    url: req.url,
    method: req.method,
    userId: req.user?.id
  });

  // Send to monitoring service
  Sentry.captureException(error);

  res.status(500).json({
    success: false,
    message: 'Internal server error',
    requestId: req.id
  });
});
```

Testing Strategy

```
Testing Pyramid:
├── Unit Tests (70%)
│   ├── Service functions
│   ├── Utility functions
│   └── Component logic
├── Integration Tests (20%)
│   ├── API endpoints
│   ├── Database operations
│   └── Payment integration
└── E2E Tests (10%)
    ├── User flows
    ├── Mobile app scenarios
    └── Cross-platform testing
```

Deployment & DevOps

```
# Docker Compose for local development
version: '3.8'
services:
  backend:
    build: ./packages/backend
    environment:
      - NODE_ENV=development
      - DATABASE_URL=postgresql://user:pass@db:5432/digibachat
    ports:
      - "3000:3000"
    depends_on:
      - db
      - redis

  db:
    image: postgres:15
    environment:
      POSTGRES_DB: digibachat
      POSTGRES_USER: user
      POSTGRES_PASSWORD: pass
    ports:
      - "5432:5432"

  redis:
    image: redis:7-alpine
    ports:
      - "6379:6379"
```

User Experience Enhancements

1. **Progressive Web App**: Offline support for basic features
2. **Dark Mode**: System preference detection and toggle
3. **Accessibility**: WCAG 2.1 compliance for inclusive design
4. **Internationalization**: Multi-language support (Hindi, English)
5. **Onboarding**: Interactive tutorials for new users
6. **Voice Commands**: Voice-based transaction recording

---

9. 🎨 UI/UX Design Guidelines

Color Palette (Trust & Finance)

```
/* Primary Colors - Light Green to Blue Aqua */
:root {
  --primary-50: #f0fdfa;   /* Lightest aqua */
  --primary-100: #ccfbf1;  /* Light mint */
  --primary-200: #99f6e4;  /* Soft aqua */
  --primary-300: #5eead4;  /* Medium mint */
  --primary-400: #2dd4bf;  /* Primary aqua */
  --primary-500: #14b8a6;  /* Main brand color */
  --primary-600: #0d9488;  /* Deeper teal */
  --primary-700: #0f766e;  /* Dark teal */
  --primary-800: #115e59;  /* Darker teal */
  --primary-900: #134e4a;  /* Darkest */

  /* Success & Danger */
  --success: #10b981;      /* Green for positive actions */
  --danger: #ef4444;       /* Red for warnings/errors */
  --warning: #f59e0b;      /* Amber for alerts */

  /* Neutrals */
  --gray-50: #f9fafb;
  --gray-100: #f3f4f6;
  --gray-500: #6b7280;
  --gray-900: #111827;
}
```

Typography & Spacing

```
/* Typography Scale */
.text-xs { font-size: 0.75rem; }    /* 12px - Captions */
.text-sm { font-size: 0.875rem; }   /* 14px - Body small */
.text-base { font-size: 1rem; }     /* 16px - Body */
.text-lg { font-size: 1.125rem; }   /* 18px - Subheadings */
.text-xl { font-size: 1.25rem; }    /* 20px - Headings */
.text-2xl { font-size: 1.5rem; }    /* 24px - Page titles */

/* Spacing Scale */
.space-1 { margin: 0.25rem; }   /* 4px */
.space-2 { margin: 0.5rem; }    /* 8px */
.space-4 { margin: 1rem; }      /* 16px */
.space-6 { margin: 1.5rem; }    /* 24px */
.space-8 { margin: 2rem; }      /* 32px */
```

---

10. 📊 Success Metrics & KPIs

Business Metrics

- **User Acquisition**: Monthly Active Users (MAU)
- **Engagement**: Average groups per user
- **Financial**: Total savings managed on platform
- **Growth**: Month-over-month user growth rate
- **Retention**: 30-day and 90-day user retention

Technical Metrics

- **Performance**: API response time < 200ms (95th percentile)
- **Availability**: 99.9% uptime SLA
- **Scalability**: Support 10,000+ concurrent users
- **Security**: Zero critical security incidents
- **Quality**: Bug report rate < 1% of active users

Financial Targets

- **Year 1**: 1,000 active groups, ₹1 crore in savings
- **Year 2**: 10,000 active groups, ₹50 crore in savings
- **Year 3**: 50,000 active groups, ₹500 crore in savings

🎉 Conclusion

This comprehensive plan provides a solid foundation for building DiGiBachat into a robust, scalable, and user-friendly group savings and lending platform. The modular architecture, security-first approach, and phased development strategy will ensure successful delivery and long-term sustainability.

Next Steps:

1. Set up development environment and monorepo structure
2. Begin Phase 1 development with core team
3. Establish CI/CD pipelines and testing frameworks
4. Start UI/UX design and user research
5. Begin regulatory compliance research