KoshPay

Digital Wallet Platform

Full-Stack Technical Deep-Dive · Mentor Review Document

Deployed on Render (Frontend + Backend + Database) · February 2026

JWT Auth + RBAC	**WebSocket Real-Time**	**Fraud Detection Engine**
Scheduled Payments	**QR Code Transfer**	**Role-Based Admin Panel**
ACID Transfers	**OTP Verification**	**Docker + Render Deploy**

Authors	Shravani Korde · Gautam Jha · Siddhant Ghodke
Organisation	GlideCloud
Stack	React 18 (Vite) · Spring Boot 3 · MySQL 8 / PostgreSQL
Deployment	Render — Frontend (static site) + Backend (web service) + Database
Security	JWT HS256 · BCrypt · RBAC (@PreAuthorize) · Fraud Engine

. Table of Contents

MENTOR PRESENTATION SUMMARY

KoshPay is a production-grade full-stack digital wallet inspired by India's UPI system. Users register, receive a unique UPI ID (e.g. **john@koshpay**), and can instantly send money, schedule future payments, scan QR codes, and manage contacts — all protected by JWT authentication, BCrypt hashing, a custom rule-based fraud engine, OTP verification, and a role-based admin panel. The entire system is deployed live on Render.

Quick Reference Numbers

Property	Value
Fraud rules	4 independent rules
BLOCK threshold	Fraud score >= 70
OTP threshold	Fraud score > 50 OR amount > Rs.1,000
Scheduler interval	Every 10 seconds (fixedRate=10000)
JWT validity	1 hour (3,600,000 ms) | HS256 signed
Password hashing	BCrypt (passwords AND PINs)
WebSocket topic	/topic/wallet/{walletId}
Test files	15 files covering all architectural layers
Admin roles	4 (SUPER_ADMIN, ANALYTICS, TRANSACTIONS, AUDIT_LOGS)
Admin seeding	Automatic on first startup via AdminInitializer
DB (local)	MySQL 8 via Docker Compose
DB (Render)	PostgreSQL via Render Managed Database

1. Project Overview & Feature Summary

KoshPay is modelled on India's Unified Payments Interface (UPI). Every user receives a unique UPI ID on registration. They can transfer money in real-time, schedule future payments with live countdown timers, scan QR codes to pre-fill transfer forms, manage a contacts list, and view full transaction history — all protected by a multi-layer security stack.

User Features

Feature	What It Does	Key Technical Detail
JWT Auth	Register and login. Role in token.	BCrypt passwords. HS256 JWT. 1-hour expiry. JwtFilter validates every request.
UPI Transfer	Send money by UPI ID.	PIN > balance > fraud > OTP > ACID DB write. @Transactional rollback on failure.
Scheduled Pay	Schedule payments for any future time.	@Scheduled every 10s. REQUIRES_NEW isolation. Frontend smart polling 5s/30s.
QR Code	Generate QR. Scan to pre-fill form.	upi://pay payload. jsQR decodes camera frames. getUserMedia API.
History	Full transaction log DEBIT/CREDIT.	Ordered by timestamp DESC. UPI IDs resolved via VPA repository.
Contacts	Save/delete UPI contacts.	Validates UPI in DB. Prevents adding self. Prefills transfer form on click.
PIN Security	Set or update 4-digit transaction PIN.	BCrypt hashed. Regex validated. Required on every transfer.
Live Balance	Balance updates within milliseconds.	STOMP WebSocket. /topic/wallet/{id}. SockJS fallback. 5s reconnect delay.

Admin Features

Tab	What It Shows	Role Access
Dashboard	Total transactions, volume, success rate, fraud block count.	Super Admin only
Analytics	Pie chart (status distribution) + Bar chart (lifecycle breakdown).	Super Admin + Analytics Admin
Transactions	All transactions, all users, with UPI IDs, amount, status.	Super Admin + Transactions Admin
Audit Logs	Every action: LOGIN, TRANSFER, FRAUD_BLOCK. Old + new balance.	Super Admin + Audit Logs Admin

2. Technology Stack

Backend

Technology	Version	Role in KoshPay
Spring Boot	3.x	Core framework. Auto-configuration, embedded Tomcat, DI container, REST layer.
Spring Security	6.x	JWT filter chain. BCrypt bean. Role-based URL protection. @EnableMethodSecurity for @PreAuthorize. CORS config.
Spring Data JPA + Hibernate	3.x/6.x	ORM. Maps Java entities to DB tables. Repository pattern. @Transactional management.
MySQL / PostgreSQL	8/15	MySQL locally via Docker. PostgreSQL on Render. InnoDB for ACID. 8 core tables.
Spring WebSocket + STOMP	3.x	Full-duplex real-time. STOMP over SockJS. enableSimpleBroker('/topic'). Pushes balance updates.
Lombok	--	@RequiredArgsConstructor, @Slf4j, @Getter/@Setter. Eliminates boilerplate.
JJWT	--	JWT generation, parsing, HS256 HMAC signing, Claims extraction.
SpringDoc / Swagger	2.x	Auto-generates OpenAPI 3. Swagger UI at /swagger-ui.html.
Docker + Compose	--	Local development. Backend (JDK 17) + MySQL container. docker compose up --build.

Frontend

Technology	Role in KoshPay
React 18 + Vite	Component-based UI. Vite for instant HMR in dev and optimised production build.
React Router v6	Client-side routing. Outlet for nested protected routes. useNavigate for navigation.
Axios	HTTP client. Request interceptor injects JWT. Response interceptor handles 401 auto-logout.
@stomp/stompjs + SockJS-client	STOMP WebSocket client. Subscribes to /topic/wallet/{walletId}. 5s auto-reconnect.
Recharts	PieChart (status distribution) + BarChart (lifecycle) in Admin Analytics.
jwt-decode	Client-side token decoding to read role claim — drives isAdmin and adminRole state.
QRCode.js	Renders UPI QR code as canvas element from upi://pay payload.
jsQR	Decodes QR from camera video frames. getUserMedia() to canvas to jsQR() to UPI ID.

3. System Architecture & Request Flow

3.1 High-Level Architecture Diagram

```
+-------------------------------------------------------------------+ | RENDER STATIC SITE | | React 18 (Vite) Frontend | | | | [Login] [Dashboard] [Transfer] [Scheduler] [Admin Panel] | | | | AuthContext (JWT decode + role) | | PrivateRoute --> user routes | | AdminRoute --> admin routes (allowedRoles check) | | AdminSidebar --> filters tabs by adminRole | +---------------------------+-----------------------------------+ | REST API (Axios + JWT Bearer token) WebSocket (STOMP over SockJS) | +---------------------------v-----------------------------------+ | RENDER WEB SERVICE | | Spring Boot 3 Backend | | | | JwtFilter --> SecurityConfig --> Controllers | | | | [AuthController] [WalletController] [AdminController] | | [UpiTransferController] [ScheduledPaymentController] | | | | Fraud Engine (4 rules) | OTP Service | Audit Logger | | WebSocket Broker (/topic/wallet/{walletId}) | | @Scheduled Executor (every 10s for scheduled payments) | +---------------------------+-----------------------------------+ | JPA / Hibernate ORM | +---------------------------v-----------------------------------+ | RENDER MANAGED DATABASE | | PostgreSQL (Render) / MySQL (local Docker) | | | | users | wallets | transactions | scheduled_payments | | audit_logs | contacts | virtual_payment_addresses | admins| +-------------------------------------------------------------------+
```

3.2 End-to-End Request Flow

```
React/Axios Database | | |-- [1] Request + Authorization: Bearer <token> ------->| | | | [2] JwtFilter.doFilterInternal() | | - Read Authorization header | | - Strip "Bearer " | | - jwtUtil.isTokenValid(token) | | - extractEmail() + extractRole() | | - Set SecurityContextHolder | | | | [3] SecurityConfig | | - Check URL vs role rules | | - /api/admin/** needs admin role | | | | [4] Controller (@PreAuthorize checked) | | - authentication.getName() = email | | | | [5] Service Layer | | - Business logic | | - PIN / Fraud / OTP checks | | - @Transactional DB writes ------------->| | | | [6] WebSocket Push (after balance change) | | - publishBalance(walletId, balance) | |<-- [7] Response JSON + WebSocket /topic/wallet/{id} --| | | | 401 response --> axios interceptor | | --> localStorage.clear() | | --> window.location.replace('/login') |
```

4. Security Layer — JWT, Filter Chain & BCrypt

4.1 JwtUtil.java — Token Generation & Validation

Method / Field	Value	What It Does
SECRET_KEY	32-char HMAC key	Keys.hmacShaKeyFor() converts to Key object for HS256 signing.
EXPIRATION_TIME	3,600,000 ms (1 hr)	Token expires 1 hour after issuedAt.
generateToken(email, role)	Signed JWT	Claims map: {role: actual_role}. Subject = email. Used for both users and admins.
extractEmail(token)	String email	getClaims().getSubject(). Used by JwtFilter to identify caller.
extractRole(token)	String role	getClaims().get('role', String.class). Builds GrantedAuthority.
isTokenValid(token)	boolean	Wraps getClaims() in try/catch. Expired or tampered token returns false.

4.2 JwtFilter — Request Interception Flow

```
Every HTTP Request | v Read 'Authorization' header | starts with 'Bearer '? | YES --+---------> strip '
Bearer ' --> raw token | | | jwtUtil.isTokenValid(token)? | | | YES -------+--------> extractEmail() +
extractRole() | | | | | Build UsernamePasswordAuthenticationToken | | (email, null,
[GrantedAuthority(role)]) | | | | | SecurityContextHolder.setAuthentication(auth) | | | NO --> skip (Spring
returns 401 at auth step) | NO ---> skip (public endpoints pass through) | v filterChain.doFilter() --
ALWAYS called regardless
```

4.3 SecurityConfig — Route Protection Rules

Route Pattern	Rule	Reason
/ws/**	permitAll()	WebSocket handshake must be public. SockJS fallback needs open endpoint.
/api/auth/**	permitAll()	Register + login must be public to obtain a token in the first place.
/api/setup/**	permitAll()	Admin initialiser — public for first-time deploy.
/swagger-ui/**, /v3/api-docs/**	permitAll()	API docs accessible without login in development.
/api/admin/**	hasAnyRole(..)	All 4 admin roles allowed. Fine control delegated to @PreAuthorize per method.
anyRequest()	authenticated()	Everything else needs a valid JWT. Missing = 401. Wrong role = 403.

■ *CSRF disabled — stateless JWT API. Form login and HTTP Basic disabled — API uses JSON body only.*

4.4 BCrypt — Password & PIN Hashing

What	On Registration	On Verification
Password	passwordEncoder.encode(rawPassword)	passwordEncoder.matches(raw, user.getPassword())
Transaction PIN	passwordEncoder.encode(rawPin)	passwordEncoder.matches(pin, user.getTransactionPin())

BCrypt is intentionally slow (configurable cost factor) and automatically salted — two identical PINs produce different hashes. The PasswordEncoder bean is defined in SecurityBeansConfig.java and injected wherever hashing or comparison is needed.

5. Role-Based Admin Panel — Complete Deep Dive

KoshPay implements a centralized role-based admin system with one Super Admin and three specialized sub-admins. Each admin sees only the tabs and API endpoints they are authorized for. Security is enforced at two independent layers: the API (Spring @PreAuthorize) and the UI (React AdminRoute + AdminSidebar). Even if someone bypasses the frontend and calls the API directly, Spring returns 403 Forbidden.

5.1 Admin Roles & Access Matrix

Role	Email (example)	Tab Access	API Access
ROLE_SUPER_ADMIN	admin@yourdomain.com	All 4 tabs	All /api/admin/** endpoints
ROLE_ANALYTICS	analytics@yourdomain.com	Analytics only	/api/admin/status-distribution
ROLE_TRANSACTIONS	transactions@yourdomain.com	Transactions only	/api/admin/transactions
ROLE_AUDIT_LOGS	auditlogs@yourdomain.com	Audit Logs only	/api/admin/audit-logs

■ *Super Admin credentials are configured via environment variables. Sub-admin emails follow the pattern role@yourdomain.com and are seeded automatically. All passwords are BCrypt-hashed before storage — never stored in plain text.*

5.2 RBAC Security Flow Diagram

```
Admin Login POST /api/auth/login | v AuthController - adminRepository.findByEmail(email) - passwordEncoder.matches(raw, admin.getPassword()) - jwtUtil.generateToken(email, admin.getRole()) <-- REAL role from DB | v JWT issued: { sub: email, role: "ROLE_ANALYTICS", exp: now+1hr } | +--------------------+--------------------------+ | | FRONTEND BACKEND | | AuthContext.jsx JwtFilter.java jwt_decode(token) extractRole(token) --> adminRole = "ROLE_ANALYTICS" --> GrantedAuthority("ROLE_ANALYTICS") --> isAdmin = true --> SecurityContextHolder.set(auth) | | AdminSidebar.jsx @PreAuthorize on each method NAV_ITEMS.filter( hasRole('SUPER_ADMIN') --> 403 item.roles.includes(adminRole)) hasAnyRole('ANALYTICS') --> 200 --> shows Analytics tab only | | RESULT: Double protection AdminRoute.jsx UI hides + API enforces allowedRoles check independently of each other --> wrong tab URL --> /admin/unauthorized | App.jsx getAdminHome() ROLE_ANALYTICS --> /admin/analytics --> no Access Denied flash on login
```

5.3 Admin Seeding — AdminInitializer

AdminInitializer runs in @PostConstruct and checks adminRepository.count() == 0 before seeding. This means restarting the server never creates duplicates. Super Admin credentials are read from environment variables ADMIN_EMAIL and ADMIN_PASSWORD. Sub-admin credentials are configured in application properties and BCrypt-hashed before storage.

```
@PostConstruct public void seedAdmins() { if (adminRepository.count() == 0) { // Super Admin -- credentials from environment variables saveAdmin(adminEmail, adminPassword, "ROLE_SUPER_ADMIN"); // Sub-admins -- fixed system accounts, credentials from config saveAdmin(analyticsEmail, analyticsPassword, "ROLE_ANALYTICS"); saveAdmin(transactionsEmail, transactionsPassword, "ROLE_TRANSACTIONS"); saveAdmin(auditLogsEmail, auditLogsPassword, "ROLE_AUDIT_LOGS"); } }
```

5.4 Per-Endpoint @PreAuthorize Rules

Endpoint	Annotation	Who Can Access
GET /api/admin/summary	hasRole('SUPER_ADMIN')	Super Admin only
GET /api/admin/status-distribution	hasAnyRole('SUPER_ADMIN','ANALYTICS')	Super Admin + Analytics Admin
GET /api/admin/transactions	hasAnyRole('SUPER_ADMIN','TRANSACTIONS')	Super Admin + Transactions Admin
GET /api/admin/audit-logs	hasAnyRole('SUPER_ADMIN','AUDIT_LOGS')	Super Admin + Audit Logs Admin

5.5 Frontend Role Filtering

AdminSidebar.jsx — Each nav item declares a roles array. The sidebar filters: NAV_ITEMS.filter(item => item.roles.includes(adminRole)). Sub-admins only see their own tab. A role badge below the logo shows which admin type is logged in.

AdminRoute.jsx — Accepts an allowedRoles prop per route. If the logged-in admin's role is not in the list they are redirected to /admin/unauthorized — not the login page, because they are authenticated, just not authorized for that specific tab.

Analytics.jsx — Checks adminRole before fetching. Super Admin calls both /admin/summary and /admin/status-distribution. Analytics Admin calls only /admin/status-distribution and derives totals from that data. The volume card is hidden.

5.6 Admin Tab Breakdown

Tab	Role Access	What It Shows
Dashboard	Super Admin only	Total transaction count, successful volume (Rs.), success rate %, fraud block count. 8 stat cards with staggered CSS animations.
Analytics	Super Admin + Analytics Admin	Pie chart (status distribution), bar chart (lifecycle). Success rate card. Volume card shown to Super Admin only.
Transactions	Super Admin + Transactions Admin	Full log: From UPI, To UPI, Amount, Status, Timestamp. Search by ID or UPI. Filter by status.
Audit Logs	Super Admin + Audit Logs Admin	Complete action trail: Login, Transfer, PIN change. Filter by action type. Old balance shown for transfer events.

6. Fraud Detection Engine — Complete Deep Dive

6.1 Architecture & Rule Pattern Diagram

```
Transfer Request | v FraudDetectionService.evaluate(FraudContext) | |-- FraudContext holds: | fromWalletId, toWalletId, amount, | userId, transactionTime, currentBalance | v for each FraudRule in List<FraudRule>: <-- Spring injects ALL @Component implementations | +-- [1] HighAmountRule.isTriggered(ctx)? --> YES: score += 70 | +-- [2] TransactionVelocityRule.isTriggered(ctx)? --> YES: score += 80 | +-- [3] WalletDrainPercentageRule.isTriggered(ctx)? --> YES: score += 80 | +-- [4] NewPayeeRule.isTriggered(ctx)? --> YES: score += 40 | v FraudResult(totalScore, decision) | score >= 70? | YES --+--> FraudDecision.BLOCK | --> throw InvalidRequestException | --> auditLog(FRAUD_BLOCK) | --> Transaction NEVER created | NO --+--> FraudDecision.ALLOW --> check OTP gate in WalletServiceImpl --> score > 50 OR amount > Rs.1,000? YES --> require OTP before proceeding NO --> proceed with transfer Open/Closed Principle: Adding a new rule = one new @Component class. Zero changes to FraudDetectionService or any existing rule.
```

6.2 All Four Rules

Rule	Trigger Condition	Score	Effect Alone
HighAmountRule	amount > Rs.10,000	70	BLOCK (70 >= 70)
TransactionVelocityRule	More than 5 txns from same wallet in last 10 minutes	80	BLOCK (80 >= 70)
WalletDrainPercentageRule	Transfer >= 80% of current wallet balance	80	BLOCK (80 >= 70)
NewPayeeRule	No prior transaction from sender to this recipient	40	ALLOW (40 < 70), OTP if amt > Rs.1k

■ *TransactionVelocityRule uses effectiveCount = DB count + 1 because the current transaction is not yet persisted when the check runs. Without +1, the 6th transaction would incorrectly see only 5 in DB and pass.*

6.3 Combined Scoring & Decision Matrix

Scenario	Rules Fired	Score	Decision	OTP?
Rs.500 to known payee	None	0	ALLOW	No
Rs.500 to new payee	NewPayee	40	ALLOW	No (40<50, amt<1k)
Rs.1,500 to known payee	None	0	ALLOW	Yes (amount > Rs.1k)
Rs.1,500 to new payee	NewPayee	40	ALLOW	Yes (amount > Rs.1k)
Rs.800 draining 82% balance	WalletDrain	80	BLOCK	N/A -- blocked first
Rs.12,000 transfer	HighAmount	70	BLOCK	N/A -- blocked first
6th txn in 10 minutes	Velocity	80	BLOCK	N/A -- blocked first
Rs.600 new payee + 6th txn	NewPayee + Velocity	120	BLOCK	N/A -- blocked first
Rs.11k to new payee	HighAmount + NewPayee	110	BLOCK	N/A -- blocked first

7. Transfer Flow — End-to-End ACID

7.1 10-Step Transfer Sequence Diagram

```
Client WalletServiceImpl (@Transactional) Database | | | |-- POST /api/transfer -------->| | {toUpiId, amount, pin, otp} | | | | | [STEP 1] | | | PIN check | | passwordEncoder.matches(pin, user.transactionPin) | | FAIL --> IllegalArgumentException | | | | [STEP 2] | | | Self-transfer check | | sender.id == toWalletId? --> FAIL | | | | [STEP 3] | | | Balance check (cheap -- before DB fraud queries) | | sender.balance < amount? --> InsufficientBalance | | | | [STEP 4] | | | Fraud engine evaluation | | score >= 70? --> BLOCK + auditLog(FRAUD_BLOCK) | | | | [STEP 5] | | | OTP gate (score>50 OR amount>Rs.1000) | | otp == null? --> return OtpResponse (HTTP 200) | | otp provided --> otpService.validateOtp() | | | | | [STEP 6] | | | Create Transaction record ---------------------->| | statusService.updateStatus(tx, INITIATED) | | | | | [STEP 7] | | | statusService.updateStatus(tx, PENDING) -------->| | | | | [STEP 8] ACID balance update | | sender.balance -= amount ---------------------->| | receiver.balance += amount ---------------------->| | | | [STEP 9] | | | Mark SUCCESS + auditLog ------------------------>| | | | | [STEP 10] | | | WebSocket push to both wallets | |<-- 200 SUCCESS ---------------| | |<== WS /topic/wallet/{id} balance update | | | | | catch(Exception e) | | | updateStatus(FAILED) ---------------------->| | | auditLog(FAILURE) | | throw e --> @Transactional ROLLBACK (wallets only) |
```

7.2 ACID Properties

Property	Mechanism	How KoshPay Achieves It
Atomicity	@Transactional + rethrow	If receiver.save() fails, sender.save() rolls back. Balance never half-updated.
Consistency	Pre-checks before any write	Balance check runs before touching DB. Self-transfer check before creating any record.
Isolation	MySQL READ_COMMITTED	Concurrent transfers from same wallet both read current balance before either commits.
Durability	MySQL InnoDB / PostgreSQL	Committed transactions survive server restart on Render.

8. Scheduled Payments System

8.1 Executor Cron Job

```
@Scheduled(fixedRate = 10000) // fires every 10 seconds public void processScheduledPayments() { // SQL: WHERE status='PENDING' AND executed=false AND scheduledAt <= NOW() List<ScheduledPayment> due = scheduledPaymentRepository.findPendingPayments(Instant.now()); if (!due.isEmpty()) log.info("Processing {} scheduled payment(s)", due.size()); for (ScheduledPayment payment : due) { processingService.executeSinglePayment(payment); // REQUIRES_NEW per payment } }
```

■ *@EnableScheduling on WalletServiceApplication activates Spring's TaskScheduler. spring.task.scheduling.pool.size=5 provides 5 worker threads so scheduled tasks don't block other async operations.*

8.2 REQUIRES_NEW Isolation Diagram

```
Executor Thread (outer transaction T0) | +-- Payment #1 --> processingService.executeSinglePayment(p1) | | | Start NEW transaction T1 (REQUIRES_NEW) | T0 is SUSPENDED | | | Validate --> Update balances --> Mark SUCCESS | Commit T1 | Resume T0 | +-- Payment #2 --> processingService.executeSinglePayment(p2) | | | Start NEW transaction T2 (REQUIRES_NEW) | | | Validate --> INSUFFICIENT BALANCE --> FAIL | Rollback T2 (only P2 affected) | Resume T0 | +-- Payment #3 --> processingService.executeSinglePayment(p3) | Start NEW transaction T3 Succeeds independently -- P2 failure had zero effect WHY THIS MATTERS: Without REQUIRES_NEW, one failure would roll back ALL payments in that executor run.
```

8.3 Frontend — Countdown Timer & Smart Polling

Feature	Implementation
Live countdown	useEffect setInterval(1000ms). getCountdown(iso) = scheduledAt - now --> d/h/m/s breakdown.
Progress bar	getProgress() = (now - createdAt) / (scheduledAt - createdAt) * 100. CSS transition animates.
Smart polling	5s interval when any PENDING payment is due within 60s. 30s otherwise. Swaps dynamically.
Edit fix	Backend: updateSchedule() sets payment.setCreatedAt(Instant.now()). Frontend: fallback uses now-5min.

9. WebSocket — Real-Time Balance Updates

9.1 Backend Config & Data Flow Diagram

```
WebSocketConfig.java @EnableWebSocketMessageBroker | +-- .addEndpoint("/ws").withSockJS() | Tries WebSocket first. | Falls back to XHR/long-poll if blocked by Render proxy. | +-- enableSimpleBroker("/topic") | In-memory broker. All /topic/** subscriptions routed here. | +-- setApplicationDestinationPrefixes("/app") Client-to-server messages to /app/** After every successful transfer: WalletServiceImpl | v BalanceWebSocketService.publishBalance(walletId, newBalance) | v messagingTemplate.convertAndSend( " /topic/wallet/" + walletId, BalanceUpdateResponse(walletId, balance) <-- serialised to JSON ) | v STOMP broker delivers to ALL subscribers of /topic/wallet/{walletId} | v React balanceSocket.js receives message --> JSON.parse(message.body) --> onBalanceUpdate(data) --> setBalance(data.balance) <-- BalanceCard re-renders instantly
```

9.2 Frontend balanceSocket.js — Key Implementation Notes

Feature	Why It Matters
Duplicate connection guard	React StrictMode can run useEffect twice. Without the active check, two sockets open and balance updates appear twice.
1500ms delay on fetch	After WS balance event, backend may not have written the transaction row yet. Delay ensures new TX appears in history.
reconnectDelay: 5000	If Render spins down briefly or network hiccups, client auto-reconnects in 5 seconds without user action.
connectHeaders with JWT	Server can validate token on WS handshake. Adds identity layer even though /ws is permitAll.

10. Frontend Architecture

Auth System

File	What It Does	Key Logic
AuthContext.jsx	Global auth state provider.	On mount: jwtDecode(token) --> extract role --> setIsAdmin, setAdminRole. ADMIN_ROLES array holds all 4 admin role strings.
PrivateRoute.jsx	Protects user routes.	loading --> spinner. !isAuthenticated --> /login. else --> render child routes.
AdminRoute.jsx	Protects admin routes by role.	Checks isAdmin + allowedRoles prop. Wrong role --> /admin/unauthorized. Not admin at all --> /dashboard.

Axios Interceptors

Interceptor	Logic	Why
Request	Read localStorage('token'). Append Authorization: Bearer token.	Every API call gets JWT automatically. No manual header setting per page.
Response	On 401: localStorage.clear() --> window.location.replace('/login').	Hard redirect resets React app. replace() prevents back button returning to protected page.

All Pages — Detailed Breakdown

Page	Route	Key Implementation
Login.jsx	/login	User/Admin toggle tabs. POST /api/auth/login --> jwtDecode --> role-aware redirect.
Dashboard.jsx	/dashboard	Fetches balance + last 5 transactions. WS connection per walletId. Balance init to null prevents Rs.0 flash.
Transfer.jsx	/transfer	Resolve UPI --> toWalletId. If response.status == ' OTP_REQUIRED' --> dynamically show OTP input. Re-submit with OTP.
ScheduledPayments.jsx	/scheduled-payments	Create/edit form. Cards with live countdown (1s tick) and progress bar. Smart 5s/30s polling.
Transactions.jsx	/transactions	Full history. DEBIT (red) / CREDIT (green). Amount sign colored. Status badge. Ordered newest first.
MyQR.jsx	/my-qr	GET /api/qr/generate --> upi://pay?pa=...&pn;=...&cu;=INR. QRCode.js renders as canvas.
ScanQR.jsx	/scan-qr	getUserMedia({video:true}) --> canvas frame --> jsQR() --> on QR detected --> navigate with UPI ID.
Contacts.jsx	/contacts	List, add by UPI+name, delete. Click contact --> prefills Transfer. UPI validated server-side.
Security.jsx	/security	PIN update form. Client validates regex before sending. PUT /api/wallet/pin.
Analytics.jsx	/admin/analytics	Role-aware fetch. Super Admin: summary + distribution. Analytics Admin: distribution only.

Page	Route	Key Implementation
AdminTransactions.jsx	/admin/transactions	Full transaction log. Search by ID or UPI handle. Filter by status. Live record count.
AuditLogs.jsx	/admin/audit-logs	Complete action trail. Filter by action type. Old balance shown for transfer events.

11. Admin Panel — APIs & Analytics

Endpoint	Returns	Role Required
GET /api/admin/summary	totalTransactions, totalVolume (BigDecimal SUCCESS sum)	SUPER_ADMIN
GET /api/admin/status-distribution	{initiated, pending, success, failed} counts	SUPER_ADMIN + ANALYTICS
GET /api/admin/transactions	All transactions, all users, full detail with UPI IDs	SUPER_ADMIN + TRANSACTIONS
GET /api/admin/audit-logs	All audit log entries across all users	SUPER_ADMIN + AUDIT_LOGS

Analytics Charts

Chart	Library	Config
Status Distribution	PieChart + Pie + Cell	outerRadius=110. 4 status slices. Success=green, Failed=red, Pending=yellow, Initiated=blue.
Lifecycle Breakdown	BarChart + Bar	CartesianGrid strokeDasharray=3 3. Rounded bar tops radius=[8,8,0,0]. Bar fill blue.
Success Rate	Calculated	(distribution.success / summary.totalTransactions * 100).toFixed(1). Shown as summary card.

Audit Log Actions

Action	Result Values	Logged When
LOGIN	SUCCESS / FAILURE	Every login attempt including wrong password
TRANSFER	SUCCESS / FAILURE / FRAUD_BLOCK	Every transfer outcome
SCHEDULED_TRANSFER	INITIATED / PENDING / SUCCESS / FAILED	Each step of scheduled payment execution

12. Database — Entities & UPI Generation

```
Entity Relationship Overview User (1) ---< Wallet (1) User has exactly one Wallet User (1) ---<
VirtualPaymentAddress (1) User has exactly one UPI ID Wallet (1) ---< Transaction (many) Wallet is
fromWallet or toWallet User (1) ---< ScheduledPayment (many) User schedules many payments User (1) ---<
AuditLog (many) Every user action logged User (1) ---< Contact (many) User saves many UPI contacts Admin (n)
-- standalone table Separate from User, has role field Transaction statuses: INITIATED --> PENDING -->
SUCCESS --> FAILED
```

Entity	Key Fields	Notes
User	id, name, email (unique), password (BCrypt), transactionPin (BCrypt), role	Has Wallet + VPA. Sender in ScheduledPayment.
Wallet	id, userId (FK), balance (DECIMAL)	Balance never goes negative due to pre-checks.
Admin	id, email (unique), password (BCrypt), role	Separate from User table. Role stored in DB, embedded in JWT.
VirtualPaymentAddress	id, upiId (unique), userId (FK), isActive	UPI format: name@koshpay. Used as receiver identifier.
Transaction	id, fromWalletId (FK), toWalletId (FK), amount, status, timestamp	Indexed by walletId + timestamp for history queries.
ScheduledPayment	id, senderId, receiverId, amount, scheduledAt, executed, createdAt, failureReason	createdAt reset on edit for correct progress bar.
AuditLog	id, userId (FK), action, result, oldBalance, newBalance, createdAt	Full forensic trail. Admin views all logs.
Contact	id, ownerId (FK), displayName, upiId, createdAt	UPI validated against VPA table on create.

UPI ID Generation Algorithm

```
UpiIdGenerator.generateBase(name): name.toLowerCase().replaceAll("[^a-z0-9]", "") // strip spaces/special
chars truncate to 15 characters --> base = "johndoe" Collision-safe loop in UserServiceImpl.createUser():
suffix = 0 do { upiId = base + (suffix == 0 ? "" : String.valueOf(suffix)) + "@koshpay" suffix++ } while
(vpaRepository.existsByUpiId(upiId)) Attempt 0: "johndoe@koshpay" --> taken? try next Attempt 1: "
johndoe1@koshpay" --> taken? try next Attempt 2: "johndoe2@koshpay" --> available --> save VPA + link to
user Minimum initial balance of Rs.1,000 enforced at registration.
```

13. Key Annotations — Complete Reference

Annotation	File	What It Does
@SpringBootApplication	WalletServiceApplication	Meta = @Configuration + @EnableAutoConfiguration + @ComponentScan. Entry point.
@EnableScheduling	WalletServiceApplication	Activates Spring's scheduling. Without this, ALL @Scheduled methods silently ignored.
@EnableMethodSecurity	SecurityConfig	Enables @PreAuthorize on individual controller methods for fine-grained RBAC.
@Scheduled(fixedRate=10000)	ScheduledPaymentExecutor	Fires every 10,000ms. fixedRate measured from start of last run (not end).
@Transactional	WalletServiceImpl + others	Wraps in DB transaction. Auto-commit on success. Rollback on RuntimeException.
@Transactional(REQUIRES_NEW)	ScheduledPaymentProcessing	Suspends outer transaction. Each payment isolated -- one failure cannot roll back others.
@Transactional(readOnly=true)	getMyBalance(), getTxHistory	Skips Hibernate dirty-checking. No write locks. Slightly faster for reads.
@PreAuthorize	AdminController	Per-method role check. Wrong role returns 403 before method body executes.
@PostConstruct	AdminInitializer	Runs after Spring context loads. Seeds admins if table is empty.
@EnableWebSocketMessageBroker	WebSocketConfig	Enables full STOMP broker. Without this, only raw WebSocket -- no /topic subscriptions.
@Service	All service classes	Spring service bean. Enables injection. Semantic: business logic layer.
@RestController	All controllers	@Controller + @ResponseBody combined. Every return auto-serialised to JSON.
@Component	JwtFilter, all FraudRules	Generic bean. FraudRule @Components auto-discovered and injected as List.
@RequiredArgsConstructor (Lombok)	Most classes	Generates constructor for all final fields. Spring uses for dependency injection.
@OncePerRequestFilter	JwtFilter extends this	Guarantees filter runs exactly once per HTTP request. Critical for JwtFilter.

14. Test Coverage — All 15 Test Files

Test File	Layer	What Is Tested
WalletServiceImplTest	Service	transfer(): valid, insufficient balance, self-transfer, wrong PIN, fraud BLOCK, OTP required, ACID rollback.
UserServiceImplTest	Service	createUser(): duplicate email, invalid PIN, success (User+Wallet+VPA). login(): correct/wrong credentials.
AuditLogServiceTest	Service	log() creates AuditLog with correct fields. Null balances handled for LOGIN events.
BalanceWebSocketServiceTest	Service	publishBalance() calls convertAndSend() with correct topic and BalanceUpdateResponse.
ContactServiceTest	Service	Add/list/delete contacts. UPI validation. Self-add prevention.
QrServiceTest	Service	QR generation returns correct upi://pay payload format.
ScheduledPaymentServiceTest	Service	Create/cancel scheduled payments. Status transitions.
ScheduledPaymentExecutorTest	Service	Executor finds due payments. Calls processingService for each.
AuthControllerTest	Controller	POST /api/auth/register: 200, 400 duplicate, 400 invalid PIN. POST /api/auth/login: 200 JWT, 401 wrong password.
WalletControllerTest	Controller	GET /api/wallet/balance: 200 auth, 401 no token. GET /api/wallet/transactions.
JwtUtilTest	Security	generateToken(). extractEmail(). extractRole(). isTokenValid(): true fresh, false expired, false tampered.
JwtFilterTest	Security	Valid token: SecurityContext populated. Missing header: empty context, chain continues. Invalid: empty.
SecurityConfigTest	Security	Public endpoints 200 no token. Protected 401. /api/admin/** 403 for ROLE_USER, 200 for admin role.
WebSocketConfigTest	Config	STOMP endpoint at /ws. SimpleBroker on /topic. App prefix /app. SockJS enabled.
ConfigTest + EntityTest + DtoTest	Config/DTO	BCryptPasswordEncoder bean. CORS config. Entity constraints. DTO serialisation. Exception HTTP status mapping.

15. Bugs Fixed & Root Cause Analysis

Bug	Symptom	Root Cause	Fix Applied
Balance flashes Rs.0	Shows Rs.0 briefly before real balance.	balance state initialized to 0. Renders before API responds.	Init balance to null. Pass balance??0 to BalanceCard only.
Stale transactions after WS	Balance updates but TX list shows old data.	fetchTransactions() called immediately. Backend still writing row.	1500ms delay before fetchTransactions() after WS event.
Scheduled stuck PENDING	Shows 'Executing...' forever.	Executor ran every 60s. No frontend polling after mount.	Executor fixedRate=10000. Frontend 5s/30s smart polling.
Progress bar broken after edit	Bar at 100% or countdown = 0 after edit.	createdAt not reset on PUT.	Backend: setCreatedAt(Instant.now()) on update. Frontend: fallback now-5min.
WS reconnects every render	Multiple connections. Repeated logs.	balance in useEffect deps. Every setBalance() triggered reconnect.	balanceRef.current tracks value. walletId-only dep array.
Sub-admin Access Denied flash	Transactions Admin lands on wrong page.	App.jsx always redirected all admins to /admin/dashboard.	getAdminHome(role) maps each role to correct landing page.
Analytics 500 for sub-admin	Analytics page crashes for ROLE_ANALYTICS.	Called /admin/summary which requires SUPER_ADMIN.	Role-aware fetch. Analytics Admin calls only /status-distribution.
Scheduler 60s delay	Payments executed up to 60s late.	fixedRate=60000 -- too slow for good UX.	Changed to fixedRate=10000. Query is idempotent -- safe to run often.

16. Render Deployment Architecture

```
GitHub Repository | | git push main | +--[auto-deploy]--+--[auto-deploy]--+ | | | v v v Render Static Render
Web Render Managed Site Service Database (React/Vite) (Spring Boot) (PostgreSQL) | | | npm run build
Dockerfile build Internal DB URL dist/ Port 8080 provided as env var VITE_API_URL DB_URL, DB_USER, to
backend service env var at DB_PASSWORD from build time Render dashboard | | | v v v https://app.
https://api. postgres://... onrender.com onrender.com .onrender.com Admin seeding: AdminInitializer runs on
first backend startup. Checks adminRepository.count() == 0. Seeds 4 admin accounts with BCrypt-hashed
passwords. No manual SQL required.
```

Environment Variables — Render Dashboard

Variable	Service	Description
ADMIN_EMAIL	Backend	Super Admin login email (configured by you)
ADMIN_PASSWORD	Backend	Super Admin login password (BCrypt-hashed before storage)
DB_HOST	Backend	PostgreSQL host from Render database connection info
DB_PORT	Backend	PostgreSQL port (default 5432)
DB_NAME	Backend	Database name from Render
DB_USER	Backend	Database username from Render
DB_PASSWORD	Backend	Database password from Render
VITE_API_URL	Frontend (build time)	https://your-backend-name.onrender.com

Single application.properties — Dual Environment Strategy

```
# One file works locally (Docker) AND on Render (env vars) # Spring syntax: ${VAR_NAME:fallback_value}
spring.datasource.url=${DB_URL:jdbc:mysql://mysql:3306/ewallet} # Local: DB_URL not set --> uses Docker
service name 'mysql' # Render: DB_URL set in dashboard --> uses Render DB connection string
spring.datasource.username=${DB_USERNAME:root} spring.datasource.password=${DB_PASSWORD:root}
spring.task.scheduling.pool.size=5 # thread pool for @Scheduled app.admin.email=${ADMIN_EMAIL}
app.admin.password=${ADMIN_PASSWORD}
```

■ *Render free tier web services spin down after 15 minutes of inactivity. When spun down, @Scheduled(fixedRate=10000) stops running -- scheduled payments due during sleep are delayed until the next request wakes the service. Solution: use UptimeRobot (free) to ping the backend every 5 minutes.*

Authors

Name	Role	Organisation
Shravani Korde	Cloud Engineer	GlideCloud
Gautam Jha	Cloud Engineer	GlideCloud
Siddhant Ghodke	Cloud Engineer	GlideCloud