**Digital Winding Engine Driver Logbook – Design Overview**

**🔐 Access & Authentication**

* **Mobile Compatibility**: Responsive design for smartphones and tablets
* **Biometric Sign-In**: Fingerprint or facial recognition for secure access
* **Role-Based Access**: Drivers, engineers, artisans, and supervisors have tailored views and permissions

**🗂️ Logbook Structure (Single Page View with Drop-downs & Columns)**

| **Section** | **Details** |
| --- | --- |
| **Shift Columns** | Separate columns for Morning, Afternoon, and Night shifts |
| **Driver Info** | Name & Surname, Biometric Signature, Start & End Time |
| **Winder Info** | Winder Name, Compartment Served |
| **Trip Tally** | Number of trips with: Persons, Material, Mineral, Explosives |
| **Final Conveyance Position** | Over & Underlay position at start and end of shift |
| **Signal Register** | Signals received and acknowledged at shift end |
| **Instructions for Relieving Driver** | Notes and handover instructions |
| **Maintenance Section** |  |

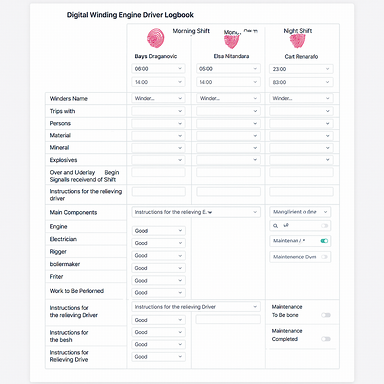
* *To Be Done*: Dropdown checklist
* *Completed*: Timestamped entries with responsible party | | **Equipment Condition Checklist** |
* Winder Motor
* Winder Drums
* Ropes
* Clutches
* Control Levers
* Safety Devices
* Signaling Systems
* Lubrication
* Illumination
* Cooling Systems
* Brake Systems
* Each with dropdown status: ✅ Good / ⚠️ Needs Attention / ❌ Faulty | | **Instructions from Specialists** |
* Engineer
* Electrician
* Rigger
* Boilermaker
* Fitter
* Shaftsman
* Artisans (Headgear/Winder)
* Each with dedicated input field or dropdown task list |

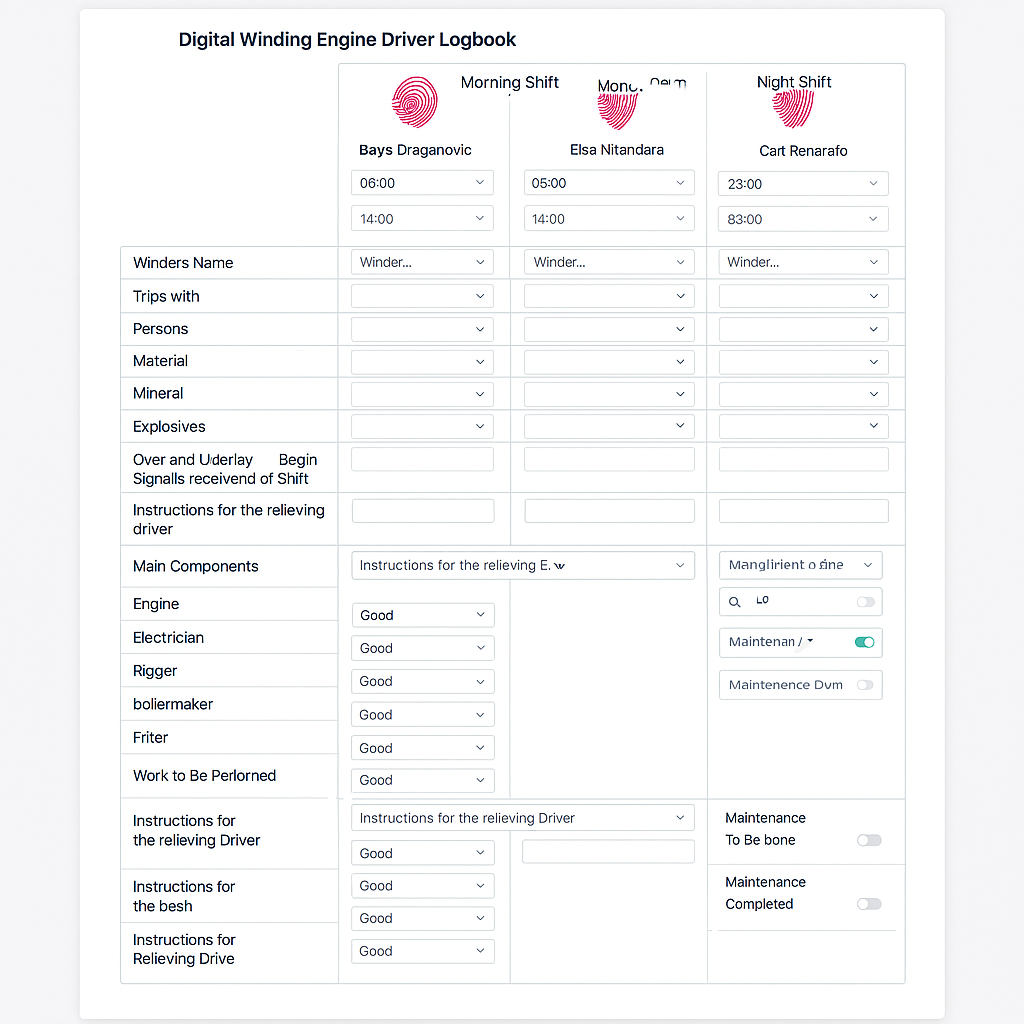
**🧠 Smart Features**

* **Auto Timestamping**: Logs start/end times automatically
* **Digital Signature Capture**: For shift handovers and maintenance confirmations
* **Color Coding**: Visual alerts for faulty equipment or overdue maintenance
* **Search & Filter**: By date, driver, winder, or issue
* **Export Options**: View summaries or reports (PDF/Excel) for audits

**🛠️ Backend & Integration**

* **Cloud-Based Storage**: Real-time syncing across devices
* **Offline Mode**: Data stored locally until connection resumes
* **Integration**: Can link with existing mine safety systems or ERP platforms





**🎯 1. Industrial Clarity (Safety & Function First)**

Ideal for mining environments where clarity and visibility are key.

| **Element** | **Color** |
| --- | --- |
| Background | Light Gray (#F5F5F5) |
| Header Text | Charcoal (#333333) |
| Shift Columns |  |

* Morning: Sky Blue (#87CEEB)
* Afternoon: Amber (#FFC107)
* Night: Slate Gray (#708090) | | Equipment Status |
* Good: Green (#4CAF50)
* Needs Attention: Orange (#FF9800)
* Faulty: Red (#F44336) | | Buttons & Drop-downs | Steel Blue (#4682B4) | | Instructions & Notes | White background with Navy text (#001F3F) |

**🌙 2. Dark Mode (Low-Light Friendly)**

Perfect for underground or night shift use.

| **Element** | **Color** |
| --- | --- |
| Background | Charcoal Black (#1C1C1C) |
| Header Text | Light Gray (#E0E0E0) |
| Shift Columns |  |

* Morning: Teal (#008080)
* Afternoon: Olive (#808000)
* Night: Indigo (#4B0082) | | Equipment Status |
* Good: Lime Green (#32CD32)
* Needs Attention: Yellow (#FFFF00)
* Faulty: Crimson (#DC143C) | | Buttons & Drop-downs | Silver (#C0C0C0) | | Instructions & Notes | Dark Gray background with White text |

**🧠 3. Smart Dashboard (Modern & Professional)**

For integration with enterprise systems or dashboards.

| **Element** | **Color** |
| --- | --- |
| Background | White (#FFFFFF) |
| Header Text | Midnight Blue (#2C3E50) |
| Shift Columns |  |

* Morning: Light Cyan (#E0FFFF)
* Afternoon: Light Coral (#F08080)
* Night: Lavender (#E6E6FA) | | Equipment Status |
* Good: Emerald (#50C878)
* Needs Attention: Gold (#FFD700)
* Faulty: Firebrick (#B22222) | | Buttons & Drop-downs | Royal Blue (#4169E1) | | Instructions & Notes | Pale Gray background with Black text |





**🧩 Interactive Elements to Incorporate**

**🔐 Biometric Sign-In**

* **Fingerprint or Face ID** integration via mobile device sensors
* Use device-native authentication (e.g., Android/iOS biometric APIs)

**📥 Drop-down Menus**

* For equipment condition: “Good,” “Needs Attention,” “Faulty”
* For shift selection, winder name, compartment served
* Use color-coded options for visual clarity (e.g., green/yellow/red)

**📝 Dynamic Input Fields**

* Auto-expandable text boxes for instructions and notes
* Character counters or tooltips for guidance
* Smart suggestions based on previous entries

**📊 Trip Tally Counters**

* Incremental buttons (+/-) for counting trips
* Auto-sum totals per category (Persons, Material, etc.)

**🧠 Conditional Logic**

* If “Faulty” is selected for a component, prompt a maintenance task
* If “Explosives” are logged, trigger a safety checklist

**🛠️ Maintenance Tracker**

* Toggle switches for “To Be Done” and “Completed”
* Timestamp and user ID capture on task completion
* Link to digital work orders or maintenance logs

**📍 Conveyance Position Tracker**

* Use sliders or numeric input fields for over/underlay positions
* Optional integration with sensor data (if available)

**📤 Signal Register**

* Dropdown for signal types + time received
* Auto-log signal confirmations with timestamp

**👥 Instruction Panels**

* Collapsible sections for each role (Engineer, Electrician, etc.)
* Predefined task templates or quick-fill buttons

**📆 Shift Scheduler**

* Calendar picker for date
* Auto-fill start/end times based on selected shift

**📷 Photo Uploads**

* Allow drivers to attach images of faulty equipment or completed maintenance
* Timestamp and location tagging for audit trail

**🧪 Bonus Features (Advanced)**

* **QR Code Scanning**: Scan shaft tags or equipment IDs
* **Voice-to-Text Input**: For hands-free logging
* **Push Notifications**: Reminders for upcoming maintenance or shift changes
* **Offline Mode**: Cache entries until reconnected
* **Trip Tally Counter** Increment/decrement buttons with an auto-sum display for Persons, Material, Mineral, Explosives.
* **Conditional Maintenance Prompt** When a component’s status is set to “Faulty,” a modal pops up to log maintenance tasks and assign to a technician.
* **Photo Upload & Annotation** Allow drivers to snap or upload images of equipment issues, then tag them with timestamps and location data.
* **Fault Status Drop-Down** A color-coded dropdown (Green/Yellow/Red) that triggers alerts or follow-up tasks behind the scenes.
* **Toggle-Based Maintenance Tracker** Swipe or tap toggles for “To Be Done” and “Completed,” auto-logging user and time.
* **Signal Register Entry** Quick-select list of signal types plus real-time timestamp, with a confirmation checkbox.

**Integrating QR Codes into Your Logbook**

**1. Generate & Affix Unique QR Labels**

* Create a list of all components (Winder Motor, Winder Drums, Ropes, etc.).
* Use a bulk-QR generator (CSV → QR images) to produce a code per component.
* Print and laminate each code, then attach it physically to its component.

**2. Data Model & Back-End Mapping**

* Assign each QR string a unique component ID in your database.
* Store metadata (component name, last inspection date, maintenance history) keyed by that ID.
* Expose an API endpoint /api/scan?qrcode=<value> that returns the matched component record.

**3. Front-End UI Changes**

**a. Add a “Scan” Column in Main Components**

| **Component** | **Status** | **Scan QR** |
| --- | --- | --- |
| Winder Motor | [▼ Good] | [🔍] |
| Winder Drums | [▼ Fair] | [🔍] |
| Ropes | [▼ Poor] | [🔍] |
| … | … | … |

* “🔍” is a tappable scan icon.
* On tap: open device camera in a modal to read the QR code.

**b. Scan Workflow**

1. User taps 🔍 next to “Winder Motor.”
2. Camera viewfinder launches; QR is detected.
3. System calls /api/scan?qrcode=WM123456.
4. Returned payload highlights the “Winder Motor” row and opens its status dropdown.
5. Optionally, pre-fill last inspection timestamp and inspector name.

**4. Example User Flow**

1. **Start Shift** → Driver signs in biometrically.
2. **Inspect Components** → Scroll to Main Components.
3. **Scan a QR** → Tap scan icon, point camera.
4. **Auto-Select Row** → “Winder Drums” row expands, shows details.
5. **Log Status** → Choose Good/Fair/Poor, add maintenance note.
6. **Submit** → Save entry with timestamp and user ID.

**5. Next Steps**

* Wireframe the scan modal and post-scan confirmation.
* Prototype with a mobile-friendly QR-scan library (e.g., ZXing or ML Kit).
* Build the back-end mapping endpoint and test end-to-end.

**QR Code–Driven Inspection & Maintenance Workflow**

**1. QR Label & Schedule Mapping**

* Assign each component (e.g., Winder Motor, Drums, Ropes… ) a unique QR code
* In your back end, map each QR → component ID → maintenance schedule:
  + Daily (Shift visual checks)
  + Weekly
  + Monthly
  + 3-Monthly (Quarterly)
  + 6-Monthly (Bi-Annual)
  + Yearly

**2. In-App Scan UI Flow**

| **Step** | **Interaction** | **Result** |
| --- | --- | --- |
| 1 | Tap 🔍 next to the component name | Launch camera viewfinder |
| 2 | Scan QR | Resolve component record via /api/scan?qrcode=<value> |
| 3 | Show “Inspection Mode” modal | Two big buttons: |

• Visual Inspection • Maintenance Routine | | 4a | Select **Visual Inspection** | Load shift-level checklist (Good/Fair/Poor + notes + sign-off) | | 4b | Select **Maintenance Routine** | Prompt to choose or auto-detect due interval (Daily/Weekly/etc.) |

**3. Visual Inspection (Shift-Level)**

* Auto-populate: • Component name & photo • Last inspection timestamp & inspector
* Checklist: • Status dropdown (Green/Yellow/Red) • Free-text “Notes” field
* Sign-off: biometric tap
* Save → Entry appears under that shift’s “Main Components” column

**4. Maintenance Routine (Artisan-Level)**

1. **Interval Selection**
   * Dropdown if multiple routines are due
   * Or auto-select based on last-completed date
2. **Task Checklist**
   * Predefined tasks per interval (e.g., lubrication points, bolt torques, rope wear measurements)
   * Checkboxes next to each task
3. **Evidence Capture**
   * Photo upload for critical checks
   * Optional voice-to-text notes
4. **Completion & Sign-Off**
   * Biometric or digital signature
   * Timestamp & user ID recorded
5. **Next Due Date**
   * System calculates & displays the next scheduled inspection

Completed tasks automatically populate the “Maintenance Completed” section and update the component’s history log.

**5. Dashboard & Alerts**

* **Due/Overdue Tracker**: at-a-glance counters per interval
* **Color-Coded Alerts**: red for overdue, amber for upcoming, green when clear
* **History View**: filter by component, date range, artisan

**6. Next Steps**

* Wireframe the **Inspection/ Maintenance** modal screens
* Prototype QR-scan using a mobile library (e.g., ZXing, ML Kit)
* Build API endpoints for schedule look-ups and logging
* Pilot with one shift and iterate on task definitions

**Daily Shaft & Winder Functions Report**

At the end of each 24-hour cycle, the system can automatically compile and deliver a comprehensive report covering every shift, every component check, and all maintenance activities. This report can be exported as PDF or Excel, emailed to stakeholders, and accessed on demand via the mobile app.

**1. Executive Summary**

* Date: 2025-09-16
* Total Trips: 120 (Persons: 75, Material: 30, Mineral: 10, Explosives: 5)
* Major Faults Logged: 2
* Maintenance Completed: 5 tasks
* Upcoming Scheduled Maintenance: 3 tasks due tomorrow

**2. Shift-by-Shift Breakdown**

| **Shift** | **Driver** | **Time** | **Trips (P/M/M/E)** | **Faults Logged** | **Maintenance Completed** | **Handover Notes** |
| --- | --- | --- | --- | --- | --- | --- |
| Morning | Bays Draganović | 06:00–14:00 | 25/10/3/1 | Winder Drums | Lubrication (Motor) | Inspect clutch wear |
| Afternoon | Elsa Nitandara | 14:00–22:00 | 30/8/4/2 | Control Levers | Brake check | Test signaling system on start |
| Night | Carl Renarro | 22:00–06:00 | 20/12/3/2 | None | Rope tensioning | Confirm illumination covers |

**3. Equipment Status Summary**

| **Component** | **Inspected Shifts** | **Good** | **Needs Attention** | **Faulty** |
| --- | --- | --- | --- | --- |
| Winder Motor | 3 | 3 | 0 | 0 |
| Winder Drums | 3 | 2 | 0 | 1 |
| Ropes | 3 | 3 | 0 | 0 |
| Clutches | 3 | 3 | 0 | 0 |
| Control Levers | 3 | 2 | 1 | 0 |
| Safety Devices | 3 | 3 | 0 | 0 |
| Signaling Systems | 3 | 3 | 0 | 0 |
| Lubrication | 3 | 2 | 1 | 0 |
| Illumination | 3 | 3 | 0 | 0 |
| Cooling Systems | 3 | 3 | 0 | 0 |
| Brake Systems | 3 | 3 | 0 | 0 |

**4. Maintenance Activities Completed**

* Motor lubrication (Morning shift)
* Brake system check (Afternoon shift)
* Rope tension adjustment (Night shift)
* Safety device calibration (Afternoon shift)
* Clutch wear inspection (Morning shift)

**5. Signals & Handover**

* Total signals received: 18
* Acknowledged end-of-shift signals: 100%
* Critical signal types: Overload warning (5), Emergency stop trigger (1)

**6. Upcoming Scheduled Maintenance**

* Daily visual inspection (all components) – due at next shift start
* Weekly full maintenance (Clutches & Drums) – due in 2 days
* Monthly lubrication audit – due in 7 days

**7. Delivery & Automation**

* **Schedule**: Auto-generate at 06:05 daily
* **Formats**: PDF (standard), Excel (data export)
* **Distribution**:
  + Email to Engineering Manager & Safety Officer
  + In-app notification to Supervisors
  + Download link on dashboard

Would you like to see a sample PDF layout or set up the automated scheduling rules and distribution list next?

can the system flag if maintenance functions is not done in regards to schedule . first to forman than later to the engineer

**Automated Overdue Maintenance Alerts & Escalation**

To ensure no maintenance task slips through the cracks, the system can continuously compare each component’s scheduled maintenance date against actual completion records. When a task becomes overdue, it automatically generates alerts—first to the foreman, then, if still unresolved, to the engineer.

**1. Overdue Detection Logic**

* Every maintenance item stores: • lastCompletedDate • intervalType (Daily/Weekly/Monthly/3-Monthly/6-Monthly/Yearly)
* System computes nextDueDate = lastCompletedDate + intervalDuration
* Daily cron job (e.g., at 00:05) scans for items where today > nextDueDate and no “completed” entry exists
* Flag status transitions:
  1. Upcoming (≤ 2 days before due)
  2. Due (today)
  3. Overdue (tomorrow onward until completed)

**2. Escalation Workflow**

| **Condition** | **Recipient** | **Channel** | **Timing** |
| --- | --- | --- | --- |
| Overdue (Day 1) | Foreman | In-app + Email | Immediately upon flagging |
| Overdue (Day 2+) | Engineer | In-app + Email + SMS | 24 hours after first alert |

1. Day 1 overdue:
   * System pushes an in-app notification to the foreman’s dashboard
   * Sends a templated email listing overdue tasks with component IDs and due dates
2. Day 2+ overdue:
   * If tasks remain uncompleted, system escalates to the engineer
   * Adds SMS or SMS-to-Email for urgent visibility

**3. Notification Content**

* Task summary: component name, interval, scheduled & overdue dates
* Assigned artisan or shift responsible
* Quick-action links: • Mark as completed • View maintenance checklist • Reassign or add notes
* Contact info for foreman or engineer for clarifications

**4. UI Indicators**

* **Dashboard Panel**
  + Red banner showing “X Overdue Maintenance Tasks”
  + Click to expand list by component and interval
* **Component Row Highlighting**
  + Overdue rows glow red in the Main Components table
  + Hover or tap shows “Due on [date]” tooltip
* **Foreman & Engineer Views**
  + Filter toggle: “Show only overdue”
  + Bulk-complete or reassign buttons

**5. Configuration & Tuning**

* Thresholds: customize how many days before due you want “Upcoming” alerts
* Escalation delays: adjust time between foreman and engineer notifications
* Channels: enable SMS, Slack, Microsoft Teams, or custom webhooks
* Roles: add safety officer or maintenance manager as additional escalation targets

**Real-Time Failure Alerts & Mobile Engineer Response**

You can turn equipment failures into instant, actionable tasks by pushing alerts to the foreman’s and engineer’s mobile devices—and empower the engineer to dispatch safe, step-by-step instructions right from their phone. Here’s how:

**Notification Mechanism**

When a component status flips to “Faulty,” the system immediately:

* Publishes an event to a notification service (e.g., Firebase Cloud Messaging, AWS SNS)
* Sends a mobile push notification to the foreman and engineer apps
* Falls back to SMS or email if the app is offline
* Embeds a deep link: tap to open the exact component alert screen

**Engineer Mobile Response Workflow**

1. Engineer taps the failure alert → opens **Component Alert Detail**.
2. Detail screen shows:
   * Component name & ID
   * Location (shaft, headgear, compartment)
   * Severity level and timestamp
   * Last maintenance record and related photos
3. Engineer enters instructions:
   * Text field for procedure steps
   * Attach photos, PDF checksheets, or safety datasheets
   * Optional voice memo for hands-free guidance
4. Engineer taps **Send Instructions**:
   * System logs timestamp, user ID, and attachments
   * Pushes follow-up notification to foreman and on-shift driver/artisan
5. Foreman and driver/artisan receive the instructions and can:
   * Acknowledge receipt (biometric sign-off)
   * Ask for clarification via in-app chat or voice note

**Key UI Elements**

* **Push Notification Card** • Icon indicating severity (red exclamation) • Component name, “Faulty” status, timestamp • “View Details” button with deep link
* **Component Alert Detail Screen** • Header: Component + Severity Badge • Status timeline (inspections, failures, pending tasks) • Instruction composer: text, attachments, voice • Send button and “Mark as Acknowledged” toggle
* **Team Chat Thread** • Contextual chat under each alert • Engineers, foreman, and drivers can collaborate

**Safety & Audit Trail**

* Every instruction and acknowledgement is timestamped and tied to the user’s biometric ID
* Versioned procedure documents ensure the correct safety protocol is followed
* Automatic escalation: if no acknowledgement within configurable window, re-alert foreman or add safety officer to the thread
* Full audit log available in the daily report

**Implementation Considerations**

* Mobile: native iOS/Android or cross-platform (React Native, Flutter) with biometric APIs
* Backend: use a pub/sub architecture for real-time events
* Offline support: cache pending alerts/instructions until online
* Security: end-to-end encryption for sensitive maintenance procedures
* Integration: link to your existing CMMS or ERP for maintenance history and work-order generation

**Hardware & Software Requirements**

**1. Hardware Requirements**

**A. Mobile Devices & Accessories**

* Rugged Tablets or Smartphones • Android (8.0+) or iOS (13.0+) with built-in camera & biometric sensor
* Protective Cases & Screen Guards
* Portable Power Banks
* Vehicle or Shaft-side Mounts

**B. QR Code Printing & Labeling**

* Industrial Label Printer (thermal or laser)
* High-durability, weather-proof labels
* Lamination equipment or heat-resistant film

**C. Network & Connectivity**

* Wi-Fi Access Points (indoor/outdoor rated)
* PoE Switches & Ethernet Cabling
* 4G/5G USB Dongles or Cellular Routers (for backup)
* VPN Gateway (for secure remote access)

**D. On-site Server & Infrastructure (Optional)**

* Rack-mount or Desktop Server (4+ cores, 16 GB RAM, SSD)
* Uninterruptible Power Supply (UPS)
* Local Storage Array or NAS (RAID 1 or RAID 5)
* Environmental Monitoring (temperature, humidity sensors)

**E. IoT & Sensor Integration (Optional)**

* Shaft Position Sensor Modules (e.g., magnetostrictive or encoder)
* Rope Tension Load Cells & Signal Amplifiers
* Environmental Probes (motor temperature, vibration)
* IoT Gateway / PLC Interface

**2. Software Requirements**

**A. Mobile Front-End**

* Cross-Platform Framework: • React Native or Flutter
* Biometric Authentication Libraries: • AndroidX Biometric API; iOS LocalAuthentication
* QR/Barcode Scanning SDKs: • ZXing, ML Kit, or Dynamsoft Barcode Reader
* Offline Data Sync: • SQLite or Realm for local caching

**B. Back-End & APIs**

* Application Server: • Node.js (Express/Koa), or .NET Core, or Spring Boot (Java)
* Database: • PostgreSQL or MySQL (relational) • MongoDB (document) optional for unstructured logs
* Real-Time Messaging: • WebSocket (Socket.io) or MQTT broker
* Push Notification Service: • Firebase Cloud Messaging (FCM) or Apple Push Notification (APNs)
* Authentication & Roles: • OAuth2 / OpenID Connect server (e.g., Keycloak)

**C. Reporting & Analytics**

* Report Engine: • JasperReports, Crystal Reports, or Pentaho
* Data Visualization: • Grafana or Kibana for dashboards
* Export Libraries: • PDFKit; Apache POI or Pandas (Excel)

**D. DevOps & Infrastructure**

* Containerization: • Docker + Docker Compose
* CI/CD Pipeline: • GitHub Actions, Jenkins, or Azure DevOps
* Infrastructure as Code: • Terraform or ARM templates (for cloud)
* Monitoring & Logging: • Prometheus + Alertmanager; ELK stack (Elastic, Logstash, Kibana)
* SSL/TLS Certificates: • Let’s Encrypt or commercial CA

**E. Optional Integrations**

* ERP/CMMS Connector: • REST or SOAP API client
* Voice-to-Text Service: • Google Cloud Speech-to-Text or Azure Speech Service
* In-App Chat / Collaboration: • Twilio Conversations or Socket.io chat module

**Next Steps**

1. **Vendor Selection**: Compare device vendors (e.g., Zebra, Samsung, Apple) and scanning SDK providers.
2. **Proof of Concept**: Stand up a minimal back-end, deploy the mobile app on 1–2 devices, and test QR-scanning + offline sync.
3. **Security Review**: Ensure biometric data, network traffic, and storage are encrypted end-to-end.
4. **Pilot Rollout**: Train a small shift team, gather feedback, then iterate before full deployment.