Phase 1: Core System & User Foundation – Granular Checklist

## 1. Electron/React UI Scaffold and Main Window Shell

### Initialize Project

* - [ ] Create project root folder on local system.
* - [ ] Initialize Git repository.
* - [ ] Add standard .gitignore (Node, Python, DB files, OS).
* - [ ] Create /frontend and /backend directories.

### Set Up Electron

* - [ ] Install Electron via npm/yarn.
* - [ ] Create main.js or main.ts Electron entry point.
* - [ ] Configure Electron to open a window on launch.
* - [ ] Set window properties (size, min size, icon).
* - [ ] Update package.json for Electron main entry and scripts (electron-dev).
* - [ ] Test Electron launches to blank page.

### Set Up React Frontend

* - [ ] Scaffold React app in /frontend (e.g., npx create-react-app .).
* - [ ] Remove React boilerplate (logo, test pages).
* - [ ] Add routing (e.g., React Router) for tab/module navigation.
* - [ ] Set up base CSS, theme provider, and global styles.
* - [ ] Create reusable layout components (Header, Sidebar, Content).
* - [ ] Commit initial React scaffold.

### Integrate Electron and React

* - [ ] Set Electron to load React’s build output (or local dev server in dev).
* - [ ] Test hot-reload and production build integration.
* - [ ] Ensure window loads the app in production (build) and development.

### Main Window & Header

* - [ ] Implement persistent header showing:
* - [ ] Mission name (placeholder value)
* - [ ] Operational period (placeholder value)
* - [ ] User role (placeholder value)
* - [ ] Add status/connection indicator (optional).
* - [ ] Commit functional main window.

### Top-Level Navigation

* - [ ] Design navigation bar with all modules as tabs:
* - [ ] Command
* - [ ] Planning
* - [ ] Operations
* - [ ] Logistics
* - [ ] Communications
* - [ ] Medical and Safety
* - [ ] Intel
* - [ ] Liaison
* - [ ] Personnel/Role Management
* - [ ] Reference Library
* - [ ] ICS Forms
* - [ ] Finance/Admin
* - [ ] Status Boards
* - [ ] Public Info
* - [ ] Settings/About
* - [ ] Wire tab switching logic (route or state).
* - [ ] Show “Coming Soon” placeholders for incomplete modules.

### Theme Support

* - [ ] Add dark and light theme support (e.g., with CSS variables or a theme provider).
* - [ ] Add UI toggle to switch between themes.
* - [ ] Test both themes for all visible components.
* - [ ] Commit UI scaffold milestone.

## 2. FastAPI Backend Integration

### Project Setup

* - [ ] Initialize FastAPI backend in /backend directory.
* - [ ] Create and activate Python virtual environment.
* - [ ] Install FastAPI, Uvicorn, and CORS middleware.
* - [ ] Create main.py entry point.
* - [ ] Set up basic FastAPI “Hello, world” root endpoint.
* - [ ] Commit initial backend structure.

### API Routing

* - [ ] Create /api prefix for all endpoints.
* - [ ] Add sample “health check” endpoint (/api/health returns status OK).
* - [ ] Enable CORS for local frontend access.
* - [ ] Test API endpoints via curl or browser.

### Frontend-Backend Integration

* - [ ] Set up frontend API base URL (local dev vs production).
* - [ ] Test frontend can fetch from /api/health.

## 3. Database Architecture: Master and Mission DBs

### Master Database

* - [ ] Create SQLite schema for master.db (tables: personnel, equipment, form templates, etc.).
* - [ ] Write initialization script to create master.db if not present.
* - [ ] Place master.db in /data directory.

### Mission Database

* - [ ] Design schema for mission-specific DBs (tables: mission metadata, logs, tasks, etc.).
* - [ ] Write script/function to create new mission DB in /data/missions/.
* - [ ] Test DB creation via API or CLI.

### API Endpoints

* - [ ] Endpoint to create a new mission DB.
* - [ ] Endpoint to select and load mission (return metadata).
* - [ ] Ensure backend logs all mission creation/selection.

### Permissions

* - [ ] App checks /data and /data/missions/ folder existence and write permissions on startup.
* - [ ] Show error or prompt if folder missing or not writable.

## 4. Role-Based Login & User Management

### Login UI

* - [ ] Make login the first screen on app launch.
* - [ ] Input for user ID (required).
* - [ ] Dropdown or autocomplete for selecting user role (roles loaded from personnel table in master.db).

### Backend Login API

* - [ ] Endpoint to validate user ID and fetch available roles.
* - [ ] Log login attempts for audit.

### App State & Restrictions

* - [ ] Store current user and role in frontend state.
* - [ ] Apply role-based restrictions to UI (disable certain tabs/menus if not authorized).
* - [ ] Log role changes to audit trail.

## 5. Mission Creation, Selection, and Loading

### Create Mission UI

* - [ ] Form to enter new mission: name, type, description, etc.
* - [ ] Validate required fields.

### List and Select Missions

* - [ ] Display list of all missions (from /data/missions/).
* - [ ] Allow user to select and load existing mission.
* - [ ] Set loaded mission as “active”; disable others.

### Header Context

* - [ ] Show active mission name/type in persistent header.
* - [ ] Grey out tabs/features if no mission loaded.

## 6. Top-Level Navigation with Module Placeholders

### Module Tabs

* - [ ] Render all planned modules as tabs (even if inactive).
* - [ ] Show module “coming soon” screen for any not implemented yet.

### Dashboard

* - [ ] Add a quick-access dashboard: widgets for active mission, operational period, critical alerts.
* - [ ] Widget placeholders can use dummy data in Phase 1.

### Dummy Status Board

* - [ ] Implement basic personnel/team status board (empty or with sample data).
* - [ ] UI to display table of users/teams/status.

## 7. System Settings & Data Folder

### Data Directory Management

* - [ ] UI in settings to show/change /data directory location.
* - [ ] App checks folder existence on startup.
* - [ ] If folder missing, prompt user to select/create before proceeding.

### Security and User Experience

### Data Storage

* - [ ] Ensure all app data (DBs, attachments) are stored locally, never to cloud.
* - [ ] Review for accidental external API calls.

### Audit Logging

* - [ ] Log all user actions: login, mission create/select, role change.
* - [ ] Store logs in a local audit table or file.

### UI Context

* - [ ] Always display current user, role, and mission somewhere in the UI.

### Phase 1 Exit Criteria

### - [ ] App launches as single exe (or from dev), main window opens.

* - [ ] User can:
* - [ ] Log in and select a role
* - [ ] Create and select a mission
* - [ ] See mission/user dashboard
* - [ ] Navigate module tabs (placeholders okay)
* - [ ] master.db and at least one mission DB are present in /data
* - [ ] No critical errors/crashes in normal use.
* - [ ] All data is saved in correct folders.

### Optional (Phase 1.5 / Stretch Goals)

### - [ ] Add About/help screen

* - [ ] Minimal user settings (theme, font size)
* - [ ] App update checker (if desired)

Phase 2: Team Operations, Personnel, and Status Boards – Granular Checklist

## 1. Personnel Roster and Org Structure

### Personnel Table & CRUD

* - [ ] Design UI for searchable/filterable personnel table.
* - [ ] Implement backend endpoints for listing personnel (read from master.db).
* - [ ] Implement "Add Personnel" dialog (name, contact info, role, status, etc.).
* - [ ] Implement "Edit Personnel" form (update name, status, assignment, etc.).
* - [ ] Implement "Remove Personnel" (soft-delete or archive entry).
* - [ ] Add search and column filter controls to roster table.

### Org Structure Management

* - [ ] Create backend data model for org units (sections, teams, branches).
* - [ ] Add UI for creating new org units (section/team/branch).
* - [ ] Implement edit/delete org units workflow.
* - [ ] Build drag-and-drop assignment of personnel to org units/teams.
* - [ ] Display interactive org chart/tree visualization.

## 2. Role Assignment, Team Creation, and Status Tracking

### Team Management

* - [ ] Build "Create Team" UI (team name, type, supervisor/leader).
* - [ ] Implement add/remove team members (from personnel list).
* - [ ] Allow edit of team attributes (type, name, leader).
* - [ ] Implement backend for saving teams to mission DB.

### Role Assignment

* - [ ] UI for assigning personnel to ICS roles (from dropdown).
* - [ ] Update personnel record with current assignment.
* - [ ] Store role and team assignment history for audit.

### Status Tracking

* - [ ] Add UI controls to set personnel/team status (Available, Assigned, Out of Service, etc.).
* - [ ] Store all status changes with timestamps.
* - [ ] Implement backend for logging assignment/status changes in mission DB.
* - [ ] Support "hot swap" reassignments, preserving previous assignment records.

## 3. Planning Module (Tasking, SITREP, Time Tracking)

### Tasking

* - [ ] Create backend data model for tasks (description, priority, status, team assignment).
* - [ ] Build UI for creating/editing tasks (form: description, priority, status, assigned teams).
* - [ ] Implement task board/table (filter by status, team, category).

### SITREP

* - [ ] Develop simple SITREP entry form (event type, summary, timestamp).
* - [ ] Display SITREP log for review/search by Planning/Command.

### Time Tracking

* - [ ] Add UI to log shift start/end for personnel and teams.
* - [ ] Store time log records in mission DB.
* - [ ] Provide ability to review/print time logs.

## 4. Operations Dashboard (Team/Assignment Views)

### - [ ] Build main Operations Dashboard view.

* - [ ] Show all teams, current assignments, and statuses.
* - [ ] Enable list, table, and (if possible) map views for teams (location can be manual entry for now).
* - [ ] Live-update UI for real-time team/personnel changes.
* - [ ] Add filters for assignment, role, team, status, or operational period.

## 5. Initial Status Boards (Team and Personnel)

### - [ ] Create Status Boards module/page.

* - [ ] Build separate boards/tabs for team status and personnel status.
* - [ ] Enable auto-update (websocket/polling) or manual refresh.
* - [ ] Add column and tag-based filters (by team, status, role, etc.).
* - [ ] Add color-coding and icons for status indicators.
* - [ ] Implement print/export to PDF/CSV.

## 6. ICS 214 Logging Interface

### - [ ] Create backend data model for ICS 214 logs (unit, entry type, text, user, timestamp).

* - [ ] Build UI for viewing and adding ICS 214 entries per team/unit and for Ops section.
* - [ ] Enable manual entry, editing, and deletion of log items.
* - [ ] Auto-log status changes, assignments, and critical actions as 214 entries.
* - [ ] Implement export/print for ICS 214 logs.

### Security and User Experience

### - [ ] Enforce role-based permissions for all sensitive UI/actions (team edits, assignments, status changes).

* - [ ] Log all personnel, team, assignment, and status changes to audit trail.
* - [ ] Ensure UI header/context always shows current user, role, and mission.

### Phase 2 Exit Criteria

### - [ ] User can view, search, and edit personnel roster (with assignments, roles, teams).

* - [ ] Teams can be created, edited, and assigned to tasks.
* - [ ] Real-time status boards (personnel and teams) are visible, filterable, and color-coded.
* - [ ] Operations dashboard displays assignments and status.
* - [ ] ICS 214 logs available for all teams and Ops section; logs exportable.
* - [ ] All changes are saved to the correct mission DB.
* - [ ] No critical errors or crashes.

### Optional (Phase 2.5 / Stretch Goals)

### - [ ] Implement drag-and-drop task assignment in dashboards.

* - [ ] Add shift/relief tracker (link to ICS 214).
* - [ ] Add map visualization for teams (if not done in Phase 1).
* - [ ] Add reminders/alerts for overdue tasks/status changes.