**Development Plan for Backend Team (FCMS)**

This **development plan** will guide the **Backend Development Team** step by step through the entire process of building the backend for the **Factory Content Management System (FCMS)**. The plan includes project setup, task allocation for each screen, testing, and build preparation, all aligned with a **10-day** deadline.

**1. Project Setup (Day 1)**

**Step 1: Set up Development Environment**

* Install **Node.js**, **Express.js**, and **MongoDB** (or SQL database, if required).
* Install necessary development tools:
  + **Visual Studio Code**
  + **Postman** for API testing
  + **Git** for version control
  + **MongoDB** (installed locally or via Docker)
* Set up a **GitHub repository** for project version control.

**Step 2: Create Basic Project Structure**

* Create a folder structure for backend code:
  + routes/: Handles routing for each screen and API.
  + controllers/: Manages business logic for each route.
  + models/: Defines MongoDB (or SQL) schema for each screen.
  + middlewares/: Contains authorization and error-handling logic.
  + utils/: Helper functions like file uploads, etc.
* Set up a **config file** for environment variables, such as database connections.

**Step 3: Install Dependencies**

* Install required dependencies:
  + express for routing.
  + socket.io for real-time communication with screens.
  + mongoose for MongoDB interactions (or relevant ORM for SQL).
  + cors, dotenv, body-parser, nodemon for development.

**Step 4: Database Setup**

* Create **MongoDB models** for each screen’s data:
  + **Work Instructions**
  + **Documents**
  + **Training**
  + **Check Sheets**
  + **Skill Matrix**
  + **Do & Don’t**
  + **Drawings**
  + **Andon**
  + **Quality**
  + **PMS (Production Management System)**
* Define relationships between data where needed (e.g., workers with skill matrix, training modules).

**Step 5: Set up WebSocket for Real-Time Communication**

* Install socket.io for real-time communication between server and client screens.
* Set up **WebSocket server** in Express to manage real-time data push for screen updates.

**2. Backend Development (Day 2 to Day 3)**

**Step 1: Work Instructions Screen Development**

* **Task: Implement Backend for Work Instructions**
  + Create routes and controllers to manage work instructions:
    - GET /work-instructions: Fetch all work instructions.
    - POST /work-instructions: Add a new work instruction.
    - PUT /work-instructions/:id: Update an existing work instruction.
    - DELETE /work-instructions/:id: Delete a work instruction.
  + Implement **WebSocket event** for real-time updates to work instructions.

**Step 2: Documents Screen Development**

* **Task: Implement Backend for Documents**
  + Create routes and controllers to manage document uploads and metadata:
    - GET /documents: Fetch all documents.
    - POST /documents: Upload a document.
    - GET /documents/:id: Fetch a specific document.
    - DELETE /documents/:id: Delete a document.
  + Implement a **search/find tool** for document retrieval.
  + Implement **documentation control** (version management, access control).

**Step 3: Training Screen Development**

* **Task: Implement Backend for Training Content**
  + Create routes and controllers to manage training content:
    - GET /training: Fetch all training modules.
    - POST /training: Add a new training module.
    - PUT /training/:id: Update an existing training module.
    - DELETE /training/:id: Delete a training module.
  + Implement **multilingual support** for training content.

**3. Backend Development (Day 4 to Day 5)**

**Step 1: Check Sheet Screen Development**

* **Task: Implement Backend for Check Sheets**
  + Create routes and controllers to manage check sheet templates:
    - GET /check-sheets: Fetch all check sheets.
    - POST /check-sheets: Add a new check sheet.
    - PUT /check-sheets/:id: Update an existing check sheet.
    - DELETE /check-sheets/:id: Delete a check sheet.
  + Implement **query-based report generation** for check sheet data.
  + Implement **event-based notifications** (e.g., when a check sheet is due for review).

**Step 2: Skill Matrix Screen Development**

* **Task: Implement Backend for Skill Matrix**
  + Create routes and controllers for managing worker skill data:
    - GET /skill-matrix: Fetch the skill matrix for workers.
    - POST /skill-matrix: Add a new skill entry.
    - PUT /skill-matrix/:id: Update a worker’s skill.
    - DELETE /skill-matrix/:id: Remove a skill entry.
  + Implement **notification system** for due dates on worker training/skill updates.

**Step 3: Drawing Screen Development**

* **Task: Implement Backend for Drawings**
  + Create routes and controllers to manage drawing files:
    - GET /drawings: Fetch all drawings.
    - POST /drawings: Upload a new drawing.
    - GET /drawings/:id: Fetch a specific drawing.
    - DELETE /drawings/:id: Delete a drawing.

**4. Backend Development (Day 6 to Day 7)**

**Step 1: Andon Screen Development**

* **Task: Implement Backend for Andon (Real-Time Notifications)**
  + Create routes and controllers for Andon-related data:
    - GET /andon: Fetch all Andon events.
    - POST /andon: Create a new Andon notification (trigger alerts).
    - PUT /andon/:id: Update an existing Andon notification.
    - DELETE /andon/:id: Remove an Andon notification.
  + Implement **real-time flow of information** using **WebSockets** for updates across all screens.

**Step 2: Quality Screen Development**

* **Task: Implement Backend for Quality (SPC Graphs)**
  + Create routes and controllers to manage SPC graphs:
    - GET /quality: Fetch quality-related data and SPC graphs.
    - POST /quality: Upload new quality data.
    - PUT /quality/:id: Update quality metrics.
    - DELETE /quality/:id: Delete quality data.
  + Implement **event-based alarms and notifications** for quality metrics.

**Step 3: PMS Screen Development**

* **Task: Implement Backend for PMS (Production Management System)**
  + Create routes and controllers to manage production scheduling:
    - GET /pms: Fetch production schedules and monitoring data.
    - POST /pms: Create a new production schedule.
    - PUT /pms/:id: Update production schedule.
    - DELETE /pms/:id: Delete production schedule.
  + Implement **real-time updates** for the PMS screen using **WebSocket** communication.

**5. Testing and Real-Time Data (Day 8)**

**Step 1: API Testing**

* Test each API endpoint using **Postman** to ensure correct functionality (CRUD operations).
* Ensure that **data validation** and **error handling** are properly set up for edge cases (e.g., missing data, invalid inputs).

**Step 2: Real-Time Communication Testing**

* Test WebSocket communication between the server and client screens for **real-time updates**.
* Ensure multiple clients can receive updates simultaneously without conflicts.

**Step 3: Load Testing**

* Simulate load on the server to test **performance** when multiple clients (displays) are connected.

**6. Finalization (Day 9 to Day 10)**

**Step 1: Final Backend Testing**

* Test the entire backend system with integrated frontend (dummy data integration).
* Perform **stress testing** for API calls and WebSocket connections.

**Step 2: Build and Deployment Preparation**

* Clean up any unused code and ensure **production-ready configurations**.
* Prepare the backend server for deployment (production server configurations, environment variables).

**Summary:**

* On **Day 1**, the Backend team will set up the environment, define the folder structure, and prepare the database models.
* From **Day 2 to Day 5**, the team will work on building the backend logic for each of the screens: **Work Instructions**, **Documents**, **Training**, **Check Sheets**, **Skill Matrix**, and **Drawings**.
* On **Day 6 to Day 7**, the team will complete backend functionality for the **Andon**, **Quality**, and **PMS** screens, integrate WebSockets for real-time updates, and test the APIs and database.
* The final **testing**, **real-time data**, and **load testing** will be done on **Day 8**, with the team preparing the server for deployment on **Day 9 to Day 10**.