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

This **development plan** will guide the **Frontend Development Team** through the entire process of building the frontend for the **Factory Content Management System (FCMS)**. The plan will include 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 necessary development tools:
  + **Visual Studio Code** (IDE)
  + **Node.js**, **npm** for package management
  + **Git** for version control
  + **Browser** (Google Chrome, Firefox) for testing
  + **Postman** for testing APIs (for integrated testing)
* Set up a **GitHub repository** for version control and collaboration.

**Step 2: Create Basic Project Structure**

* Create a folder structure for frontend code:
  + src/: Contains all frontend code.
  + components/: Reusable UI components (buttons, forms, etc.).
  + screens/: Specific screen views like **Work Instructions**, **Documents**, etc.
  + assets/: Static assets like images, videos, icons, etc.
  + styles/: Global CSS styles and theming.
  + utils/: Helper functions like API calls, socket.io communication.
* Set up **React project** using **Create React App** or **Next.js** (for server-side rendering if necessary).
* Set up **CSS framework** for UI components (e.g., **Bootstrap**, **Material UI**, or custom CSS).

**Step 3: Install Dependencies**

* Install required dependencies:
  + **axios** for making API calls to the backend.
  + **socket.io-client** for real-time communication.
  + **react-router-dom** for routing between screens.
  + **redux** or **context API** for state management (if required).
  + **react-i18next** for multilingual support.

**2. Frontend Development (Day 2 to Day 4)**

**Step 1: Work Instructions Screen Development**

* **Task: Build UI for Work Instructions Screen**
  + Create the layout to display work instructions with multiple formats (JPEG, MP4, PDF).
  + Implement components to fetch work instructions from the server:
    - Display text, images, and videos.
    - Implement pagination or infinite scroll if needed.
    - Use **axios** to call the backend API for fetching work instructions.
  + Implement **multilingual support** for content (text translation).
  + Integrate **real-time updates** via **WebSocket** to push content changes (e.g., new work instructions).

**Step 2: Documents Screen Development**

* **Task: Build UI for Documents Screen**
  + Create a layout to display documents (e.g., PDFs, images, etc.).
  + Implement the search/find tool:
    - Display document metadata (name, type, version, etc.).
    - Allow searching by name or document type.
  + Integrate **documentation control** features:
    - Support for multiple document versions.
    - Allow users to download or view documents.
  + Implement **real-time updates** using WebSockets for document changes.

**Step 3: Training Screen Development**

* **Task: Build UI for Training Screen**
  + Create a layout to display training modules (titles, descriptions, and content).
  + Implement configurable sections for training content.
  + Fetch training content from the backend using **axios**.
  + Implement **multilingual support** for training content.
  + Implement **real-time updates** using WebSockets for content changes.

**3. Frontend Development (Day 5 to Day 7)**

**Step 1: Check Sheet Screen Development**

* **Task: Build UI for Check Sheet Screen**
  + Design the layout for displaying standard and customized check sheets.
  + Allow users to view, fill out, and submit check sheets.
  + Implement **multilingual support** and **image display** for check sheets.
  + Implement **query-based reports** for check sheet data and provide download options.
  + Implement **event-based notifications** for due dates or reminders.

**Step 2: Skill Matrix Screen Development**

* **Task: Build UI for Skill Matrix Screen**
  + Design the layout for displaying worker skill data.
  + Display details like worker names, skills, training completion status, etc.
  + Implement **notification system** to show alerts about skill updates or due dates.
  + Fetch skill matrix data from the backend using **axios**.
  + Implement **real-time updates** to reflect changes to the skill matrix in real time.

**Step 3: Drawing Screen Development**

* **Task: Build UI for Drawing Screen**
  + Design a UI that supports viewing drawings in different formats (JPEG, PNG, etc.).
  + Implement zoom in/out and pan functionality for drawings.
  + Fetch drawing data from the server using **axios**.
  + Implement **real-time updates** to display newly uploaded drawings or changes.

**4. Frontend Development (Day 8 to Day 9)**

**Step 1: Andon Screen Development**

* **Task: Build UI for Andon Screen**
  + Create a UI that displays real-time production flow data.
  + Implement **multiple communication methods** (visual, SMS, email) for Andon events.
  + Implement **escalation matrix** to manage how issues are escalated.
  + Fetch Andon event data from the backend using **axios**.
  + Implement **real-time updates** via **WebSockets** for Andon data.

**Step 2: Quality Screen Development**

* **Task: Build UI for Quality Screen**
  + Design the UI to display digital SPC (Statistical Process Control) graphs.
  + Display relevant quality data and metrics.
  + Implement **data retention** and provide **download** functionality for SPC data.
  + Implement **event-based alarms** to notify users when certain quality thresholds are met.

**Step 3: PMS Screen Development**

* **Task: Build UI for PMS (Production Management System) Screen**
  + Create a layout to display real-time production data and scheduling information.
  + Implement graphical presentation of production metrics (charts, graphs).
  + Fetch PMS data from the backend using **axios**.
  + Implement **real-time updates** via **WebSockets** for production data changes.

**5. Testing and Finalization (Day 9 to Day 10)**

**Step 1: API Integration Testing**

* Test each screen's API integration by fetching data from the backend and displaying it correctly.
* Ensure all UI components are connected to the backend and the data is displayed as expected.
* Check for edge cases and handle errors gracefully (e.g., when no data is available, API errors).

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

* Test **WebSocket connections** between the frontend and backend to ensure real-time updates are properly displayed on the screens.
* Ensure multiple clients (displays) receive data updates simultaneously without conflicts.

**Step 3: Cross-Browser Testing**

* Test the application on different browsers (Google Chrome, Firefox, etc.) to ensure consistency in design and functionality.
* Ensure **responsive design** for mobile and tablet compatibility (if applicable).

**Step 4: Build and Deployment Preparation**

* Minimize CSS and JavaScript for optimal performance.
* Use **Webpack** or **Vite** for bundling the frontend code.
* Create a **production build** of the React app using the npm run build command.
* Ensure the app is ready for deployment.

**Step 5: Demo Preparation (Day 10)**

* Prepare a demo of the completed frontend screens to share with the client.
* Ensure each screen is functional and all real-time updates are working as expected.
* Verify that all necessary features and functionalities are implemented.

**Summary:**

* On **Day 1**, the frontend team will set up the environment, install dependencies, and prepare the folder structure.
* From **Day 2 to Day 4**, the team will work on building the frontend UI for the **Work Instructions**, **Documents**, and **Training** screens.
* From **Day 5 to Day 7**, the team will focus on **Check Sheets**, **Skill Matrix**, and **Drawing** screens.
* On **Day 8 to Day 9**, the team will implement the **Andon**, **Quality**, and **PMS** screens, perform API integration testing, and prepare for the final demo.
* On **Day 9 to Day 10**, the frontend team will perform **cross-browser testing**, prepare the **production build**, and finalize the project for the client demo.