Angular IoT Dashboard - Complete Development Guide

Table of Contents

Project Overview

This Angular application is a **Industrial IoT Control Center Dashboard** that provides real-time monitoring of industrial sensors, systems, and alerts. The application demonstrates modern Angular development practices using standalone components, signals, and advanced data visualization.

Key Features

- **Real-time sensor monitoring** with 6 different sensor types
- **Interactive charts and gauges** using ngx-echarts library
- **System alerts management** with real-time notifications
- **Dark/Light theme switching** with persistent storage
- **Responsive design** for desktop, tablet, and mobile
- **Professional industrial UI** with glassmorphism effects

Architecture & Project Structure

```
src/app/
├── app.ts                          # Main application component
├── app.config.ts                   # Application configuration
├── app.routes.ts                   # Routing configuration
├── app.scss                        # Global styles
├── core/                           # Core application services
│   └── services/
│       └── theme.service.ts        # Theme management
└── features/                       # Feature modules
    └── iot/                        # IoT dashboard feature
        ├── components/             # Feature components
        │   ├── iot-dashboard.component.ts      # Main dashboard
        │   ├── sensor-card.component.ts        # Individual sensor cards
        │   └── alert-panel.component.ts        # System alerts panel
        ├── services/               # Feature services
        │   └── iot-data.service.ts             # Data management
        └── models/                 # Data models
            └── sensor.model.ts                 # Type definitions
```

Architecture Principles

- **Feature-based organization**: Code organized by business features
- **Standalone components**: Using Angular's latest component architecture
- **Reactive programming**: Leveraging RxJS and Angular Signals
- **Separation of concerns**: Clear division between UI, business logic, and data

---

Core Technologies

Angular Framework (v17+)

- **Standalone Components**: Modern component architecture without NgModules
- **Angular Signals**: Reactive state management
- **Control Flow Syntax**: `@if`, `@for`, `@empty` template syntax
- **Dependency Injection**: Service-based architecture

Chart Visualization

- **ngx-echarts**: Angular wrapper for Apache ECharts
- **ECharts**: Powerful charting library with extensive customization
- **Chart Types**: Line charts, bar charts, gauge charts, pie charts

Styling & UI

- **SCSS**: Advanced CSS preprocessing
- **CSS Custom Properties**: Dynamic theming support
- **Flexbox & CSS Grid**: Modern layout techniques

- **Glassmorphism**: Modern UI design pattern with backdrop filters

Setup & Installation

Prerequisites

```
# Required versions
Node.js: 18.x or higher
npm: 9.x or higher
Angular CLI: 17.x or higher
```

Installation Steps

```
# 1. Clone the repository
git clone <repository-url>
cd todo-app

# 2. Install dependencies
npm install

# 3. Install chart dependencies
npm install ngx-echarts echarts

# 4. Start development server
npm start

# 5. Open browser
http://localhost:4200
```

Development Commands

```
# Build for production
npm run build

# Run tests
npm test

# Lint code
npm run lint

# Format code
npm run format
```

Component Architecture

1. Main Application Component (`app.ts`)

Purpose: Root component that provides navigation and theme management.

Key Features:

- Theme toggle functionality
- Navigation to IoT dashboard
- Global layout structure

```
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [CommonModule, RouterOutlet, RouterLink],
  template: `
    <div class="app-container" [class.light-theme]="!themeService.isDarkMode()">
      <main>
        <router-outlet />
      </main>
    </div>
  `
})
export class AppComponent {
  readonly themeService = inject(ThemeService);
}
```

2. IoT Dashboard Component (`iot-dashboard.component.ts`)

Purpose: Main dashboard container that orchestrates sensor cards and alerts.

Key Responsibilities:

- Layout management with CSS Grid
- Data coordination between components
- Responsive design implementation

```
@Component({
  selector: 'app-iot-dashboard',
  standalone: true,
  imports: [CommonModule, SensorCardComponent, AlertPanelComponent],
  template: `
    <div class="industrial-dashboard">
      <div class="header">
        <h1>Industrial Control Center</h1>
      </div>

      <div class="grid">
        @for (sensor of iotService.sensorsData(); track sensor.id) {
          <app-sensor-card
            [sensorData]="sensor"
            (controlClick)="onControlClick($event)"
          />
        }
      </div>

      <div class="alert-section">
        <app-alert-panel [alerts]="iotService.alertsData()" />
      </div>
    </div>
  `
})
```

CSS Grid Layout:

```
.grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(350px, 1fr));
  gap: 30px;
  max-width: 1300px;
  margin: 0 auto;
  align-items: start;
}
```

3. Sensor Card Component (`sensor-card.component.ts`)

Purpose: Individual sensor display with charts and controls.

Key Features:

- Dynamic chart type selection (gauge, line, bar, pie)
- Real-time data visualization
- Interactive controls
- Responsive design

Chart Type Logic:

```
shouldShowGauge(): boolean {
  const type = this.sensorData().type;
  return type === SensorType.TEMPERATURE || type === SensorType.PRESSURE;
}

shouldShowChart(): boolean {
  const type = this.sensorData().type;
  return type === SensorType.HUMIDITY ||
         type === SensorType.MOTOR_SPEED ||
         type === SensorType.PRODUCTION ||
         type === SensorType.ENERGY;
}
```

4. Alert Panel Component (`alert-panel.component.ts`)

Purpose: System alerts and notifications display.

Features:

- Real-time alert updates
- Color-coded alert types (critical, warning, info)
- Scrollable alert list
- Time formatting

Service Layer

1. IoT Data Service (`iot-data.service.ts`)

Purpose: Centralized data management for sensors and alerts.

Key Responsibilities:

- Mock data generation for 6 sensor types
- Real-time data simulation
- State management using Angular Signals
- Control updates

```
@Injectable({
  providedIn: 'root'
})
export class IotDataService {
  private readonly _sensorsData = signal<SensorData[]>([]);
  private readonly _alertsData = signal<Alert[]>([]);

  readonly sensorsData = this._sensorsData.asReadonly();
  readonly alertsData = this._alertsData.asReadonly();

  constructor() {
    this.initializeSensors();
    this.initializeAlerts();
    this.startRealTimeUpdates();
  }

  private startRealTimeUpdates(): void {
    interval(2000).subscribe(() => {
      this.updateSensorValues();
    });
  }
}
```

2. Theme Service (`theme.service.ts`)

Purpose: Theme management with persistence.

Features:

- Dark/Light mode switching
- localStorage persistence
- System preference detection
- Reactive theme state

```
@Injectable({
  providedIn: 'root'
})
export class ThemeService {
  private readonly _isDarkMode = signal<boolean>(true);
  readonly isDarkMode = this._isDarkMode.asReadonly();

  constructor() {
    this.loadThemeFromStorage();
    this.applyTheme();
  }

  toggleTheme(): void {
    const newTheme = !this._isDarkMode();
    this._isDarkMode.set(newTheme);
    this.saveThemeToStorage();
    this.applyTheme();
  }
}
```

Data Models

Sensor Data Model (`sensor.model.ts`)

Purpose: Type definitions for sensor data and system alerts.

```
export interface SensorData {
  id: string;
  name: string;
  type: SensorType;
  currentValue: number;
  unit: string;
  progress: number;
  icon: string;
  color: {
    primary: string;
    secondary?: string;
  };
  chartData?: number[];
  controls?: SensorControl[];
  status: 'active' | 'inactive' | 'warning' | 'error';
}

export enum SensorType {
  TEMPERATURE = 'temperature',
  HUMIDITY = 'humidity',
  PRESSURE = 'pressure',
  MOTOR_SPEED = 'motor_speed',
  ENERGY = 'energy',
  PRODUCTION = 'production'
}

export interface Alert {
  id: string;
  type: 'warning' | 'error' | 'info' | 'critical';
  title: string;
  icon: string;
  timestamp: Date;
}
```

Chart Integration (ngx-echarts)

Installation & Configuration

```
npm install ngx-echarts echarts
```

App Configuration (`app.config.ts`):

```
import { provideEcharts } from 'ngx-echarts';

export const appConfig: ApplicationConfig = {
  providers: [
    provideEcharts(),
    // other providers
  ]
};
```

Chart Types Implementation

1. Gauge Charts (Temperature & Pressure)

```
readonly gaugeOptions = computed((): EChartsOption => {
  const sensor = this.sensorData();
  const value = sensor.currentValue;
  const maxValue = this.getMaxValueForSensor(sensor.type);

  return {
    series: [{
      type: 'gauge',
      startAngle: 200,
      endAngle: -40,
      min: 0,
      max: maxValue,
      progress: {
        show: true,
        roundCap: true,
        width: 8
      },
      itemStyle: {
        color: sensor.color.primary,
        shadowColor: sensor.color.primary + '40',
        shadowBlur: 10
      },
      data: [{
        value: value,
        name: sensor.name
      }]
    }]
  };
});
```

2. Line Charts (Humidity & Motor Speed)

```
// Line chart with area fill and smooth curves
series: [{
  type: 'line',
  data: chartData,
  smooth: true,
  symbol: 'circle',
  symbolSize: 4,
  lineStyle: {
    color: sensor.color.primary,
    width: 2,
  },
  areaStyle: {
    color: {
      type: 'linear',
      colorStops: [
        { offset: 0, color: sensor.color.primary + '40' },
        { offset: 1, color: sensor.color.primary + '00' }
      ]
    }
  }
}]
```

3. Bar Charts (Production)

```
// Gradient bar chart with rounded corners
series: [{
  type: 'bar',
  data: chartData,
  itemStyle: {
    color: {
      type: 'linear',
      colorStops: [
        { offset: 0, color: sensor.color.primary },
        { offset: 1, color: sensor.color.secondary }
      ]
    },
    borderRadius: [2, 2, 0, 0]
  }
}]
```

4. Pie/Donut Charts (Energy)

```
// Energy usage donut chart
series: [{
  type: 'pie',
  radius: ['40%', '70%'],
  data: [
    {
      value: usedEnergy,
      name: 'Used',
      itemStyle: { color: sensor.color.primary }
    },
    {
      value: remainingEnergy,
      name: 'Available',
      itemStyle: { color: 'rgba(255,255,255,0.1)' }
    }
  ]
}]
```

Theme System

Implementation Strategy

CSS Custom Properties for dynamic theming:

```
:root {
  --bg-primary: #0a0f1c;
  --bg-secondary: #1a1f2e;
  --text-primary: #ffffff;
  --accent-color: #64ffda;
}

.light-theme {
  --bg-primary: #ffffff;
  --bg-secondary: #f5f5f5;
  --text-primary: #333333;
  --accent-color: #1976d2;
}
```

Service Integration:

```
private applyTheme(): void {
  const theme = this._isDarkMode() ? 'dark' : 'light';
  document.documentElement.setAttribute('data-theme', theme);

  if (theme === 'light') {
    document.body.classList.add('light-theme');
  } else {
    document.body.classList.remove('light-theme');
  }
}
```

Responsive Design

Breakpoint Strategy

```
// Mobile First Approach
@media (max-width: 768px) {
  .grid {
    grid-template-columns: 1fr;
    gap: 20px;
  }

  .card {
    padding: 20px;
    height: 320px;
  }
}

@media (max-width: 1200px) {
  .grid {
    grid-template-columns: repeat(auto-fit, minmax(320px, 1fr));
    gap: 25px;
  }
}
```

Key Responsive Features

1. **Flexible Grid Layout**: Auto-fit columns that adapt to screen size
2. **Scalable Typography**: Responsive font sizes using rem units
3. **Adaptive Spacing**: Dynamic padding and margins
4. **Touch-Friendly Controls**: Larger buttons on mobile devices
5. **Optimized Chart Heights**: Smaller charts on mobile screens

---

Key Implementation Details

1. Angular Signals Usage

Benefits:

- Better performance than traditional observables for simple state
- Automatic change detection
- Simplified syntax

```
// Service
private readonly _sensorsData = signal<SensorData[]>([]);
readonly sensorsData = this._sensorsData.asReadonly();

// Component
readonly chartOptions = computed(() => {
  const sensor = this.sensorData();
  return this.generateChartConfig(sensor);
});
```

2. Standalone Components

Advantages:

- Reduced bundle size
- Clearer dependencies
- Better tree-shaking

```
@Component({
  selector: 'app-sensor-card',
  standalone: true,
  imports: [CommonModule, NgxEchartsDirective],
  // component definition
})
```

3. Modern Control Flow

New Angular Syntax:

```
<!-- Conditional rendering -->
@if (shouldShowGauge()) {
  <div class="gauge-container">
    <!-- gauge content -->
  </div>
} @else if (shouldShowChart()) {
  <div class="chart-container">
    <!-- chart content -->
  </div>
}

<!-- List rendering -->
@for (sensor of sensors(); track sensor.id) {
  <app-sensor-card [sensorData]="sensor" />
} @empty {
  <div>No sensors available</div>
}
```

Best Practices Used

1. Code Organization

- Feature-based folder structure
- Single responsibility principle
- Clear separation of concerns

2. Performance Optimization

- OnPush change detection strategy
- Lazy loading of features
- Optimized chart rendering
- Minimal re-renders with signals

3. Type Safety

- Comprehensive TypeScript interfaces
- Strict type checking enabled
- Generic type usage where appropriate

4. Accessibility

- Semantic HTML structure
- ARIA labels for interactive elements
- Keyboard navigation support
- Color contrast compliance

5. Error Handling

- Try-catch blocks for async operations
- Graceful fallbacks for missing data
- User-friendly error messages

Troubleshooting

Common Issues & Solutions

1. Charts Not Rendering

Problem: ngx-echarts charts appear blank or don't load

Solution:

```
// Ensure ECharts is properly imported in app.config.ts
import { provideEcharts } from 'ngx-echarts';

// Check that the chart container has proper dimensions
.chart-container {
  width: 100%;
  height: 120px; // Fixed height required
}
```

2. Theme Not Persisting

Problem: Theme resets on page refresh

Solution:

```
// Verify localStorage is being used correctly
private saveThemeToStorage(): void {
  localStorage.setItem('theme', this._isDarkMode() ? 'dark' : 'light');
}

private loadThemeFromStorage(): void {
  const savedTheme = localStorage.getItem('theme');
  if (savedTheme) {
    this._isDarkMode.set(savedTheme === 'dark');
  }
}
```

3. Mobile Layout Issues

Problem: Content cut off or overlapping on mobile

Solution:

```
// Ensure proper viewport handling
.industrial-dashboard {
  width: 100%;
  max-width: 100vw;
  box-sizing: border-box;
  overflow-x: hidden;
}

// Use relative units for responsiveness
@media (max-width: 768px) {
  .card {
    min-width: auto;
    max-width: 100%;
  }
}
```

4. Real-time Updates Not Working

Problem: Sensor values don't update automatically

Solution:

```
// Verify interval subscription is active
private startRealTimeUpdates(): void {
  interval(2000).subscribe(() => {
    this.updateSensorValues();
  });
}

// Ensure signals are being updated correctly
private updateSensorValues(): void {
  const updatedSensors = this._sensorsData().map(sensor => ({
    ...sensor,
    currentValue: this.generateRandomValue(sensor.type)
  }));
  this._sensorsData.set(updatedSensors);
}
```

Conclusion

This Angular IoT Dashboard demonstrates modern web development practices with:

- **Modern Angular Features**: Standalone components, signals, and new control flow
- **Professional UI**: Industrial design with glassmorphism effects
- **Data Visualization**: Advanced charts using ngx-echarts
- **Responsive Design**: Mobile-first approach with flexible layouts
- **Performance**: Optimized rendering and minimal re-renders
- **Maintainability**: Clean architecture and separation of concerns

The application serves as an excellent foundation for building industrial monitoring systems or can be adapted for other real-time dashboard requirements.

Additional Resources

- Angular Documentation (https://angular.io/docs)
- ngx-echarts Documentation (https://github.com/xieziyu/ngx-echarts)
- ECharts Examples (https://echarts.apache.org/examples/)
- CSS Grid Guide (https://css-tricks.com/snippets/css/complete-guide-grid/)
- Angular Signals Guide (https://angular.io/guide/signals)

Document Version: 1.0
Last Updated: September 24, 2025
Author: Angular Development Team