Enhanced ODK MCP System: Local Development Setup Guide

Author: Manus AI
Version: 2.0
Date: December 2024
Document Type: Technical Setup Guide

Table of Contents

Introduction

The Enhanced ODK MCP System represents a comprehensive data collection and analysis platform designed specifically for NGOs, think tanks, CSR organizations, and research institutions. This local development setup guide provides detailed instructions for setting up the complete system on your local desktop environment, including both the web application and Android mobile application components.

The system architecture follows a microservices approach with Model Context Protocol (MCP) servers handling different aspects of data management, AI-powered analytics providing intelligent insights, and a modern React-based frontend offering an intuitive user experience. The mobile application, built with React Native, provides offline-first data collection capabilities with automatic synchronization when connectivity is restored.

This guide assumes you have basic familiarity with software development concepts and command-line operations. Each section provides step-by-step instructions with detailed explanations to ensure successful setup regardless of your experience level. The setup process typically takes 2-3 hours for a complete installation, depending on your system specifications and internet connection speed.

The local development environment mirrors the production architecture, allowing you to test all features including the subscription system, AI analytics, cross-project comparisons, and administrative functions. This ensures that any modifications or customizations you make will work seamlessly when deployed to production environments.

System Requirements

Hardware Requirements

The Enhanced ODK MCP System requires adequate hardware resources to run all components simultaneously during development. The minimum and recommended specifications ensure optimal performance across all system components.

Minimum Requirements: - **Processor:** Intel Core i5 or AMD Ryzen 5 (4 cores, 2.5GHz) - **Memory:** 8GB RAM (12GB recommended for optimal performance) - **Storage:** 50GB available disk space (SSD recommended) - **Network:** Stable internet connection for package downloads and API testing

Recommended Requirements: - **Processor:** Intel Core i7 or AMD Ryzen 7 (8 cores, 3.0GHz or higher) - **Memory:** 16GB RAM or higher - **Storage:** 100GB available SSD storage - **Graphics:** Dedicated GPU for enhanced data visualization rendering - **Network:** High-speed broadband connection (25+ Mbps)

The system utilizes multiple concurrent processes including PostgreSQL database server, Redis cache, multiple Flask microservices, React development server, and React Native Metro bundler. Adequate RAM ensures smooth operation without system slowdowns, while SSD storage significantly improves database performance and application startup times.

Software Requirements

Operating System Support: - **Windows:** Windows 10 (version 1903 or later) or Windows 11 - **macOS:** macOS 10.15 Catalina or later (macOS 12 Monterey recommended) - **Linux:** Ubuntu 20.04 LTS, Ubuntu 22.04 LTS, or equivalent distributions

Required Software Components: - **Node.js:** Version 18.x or 20.x LTS (includes npm package manager) - **Python:** Version 3.9, 3.10, or 3.11 (3.11 recommended for optimal performance) - **PostgreSQL:** Version 13, 14, or 15 (version 15 recommended) - **Redis:** Version 6.x or 7.x for caching and session management - **Git:** Latest version for source code management

Development Tools: - **Code Editor:** Visual Studio Code, WebStorm, or similar with React/Python extensions - **API Testing:** Postman, Insomnia, or similar REST client - **Database Management:** pgAdmin, DBeaver, or command-line psql client - **Android Development:** Android Studio (for mobile app development and testing)

Browser Requirements: - **Primary:** Google Chrome (version 100+) or Mozilla Firefox (version 100+) - **Testing:** Safari (macOS), Microsoft Edge for cross-browser compatibility - **Mobile Testing:** Chrome DevTools mobile emulation or physical devices

The system has been extensively tested across these platforms to ensure consistent behavior and performance. While other software versions may work, the specified versions have been validated for compatibility and optimal performance.

Environment Setup

Initial System Preparation

Before beginning the installation process, ensure your development environment is properly configured with all necessary prerequisites. This section provides detailed instructions for each supported operating system, with specific commands and configuration steps.

Windows Setup Process:

Windows users should begin by enabling Windows Subsystem for Linux (WSL2) for optimal compatibility with the development tools. Open PowerShell as Administrator and execute the following commands:

```
 # Enable WSL2
wsl --install

# Install Windows Terminal for better command-line experience
winget install Microsoft.WindowsTerminal

# Install Git for Windows
winget install Git.Git
```

```
# Install Node.js LTS version
winget install OpenJS.NodeJS.LTS

# Install Python 3.11
winget install Python.Python.3.11
```

After installing WSL2, restart your computer and set up Ubuntu within WSL2. This provides a Linux environment that closely matches production servers, reducing deployment issues and ensuring consistent behavior across development and production environments.

macOS Setup Process:

macOS users should install Homebrew package manager first, which simplifies the installation of development tools and maintains consistent versions across team members:

```
 # Install Homebrew
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/
Homebrew/install/HEAD/install.sh)"

# Update Homebrew and install essential tools
brew update
brew install git node python@3.11 postgresql@15 redis

# Install additional development tools
brew install --cask visual-studio-code android-studio
```

The Homebrew installation automatically handles dependency management and ensures all components are properly linked. This approach minimizes configuration issues and provides easy updates for all development tools.

Linux (Ubuntu) Setup Process:

Ubuntu users can utilize the built-in package manager with additional repositories for the latest versions of development tools:

```
 # Update system packages
sudo apt update && sudo apt upgrade -y

# Install essential development tools
sudo apt install -y curl wget git build-essential

# Install Node.js 20.x LTS
curl -fsSL https://deb.nodesource.com/setup_20.x | sudo -E bash
-
```

```
sudo apt install -y nodejs

# Install Python 3.11 and development headers
sudo apt install -y python3.11 python3.11-dev python3.11-venv
python3-pip

# Install PostgreSQL 15
sudo apt install -y postgresql-15 postgresql-contrib-15

# Install Redis server
sudo apt install -y redis-server

# Install Android development tools
sudo snap install android-studio --classic
```

Environment Variables Configuration

Proper environment variable configuration ensures secure and flexible system operation across different deployment scenarios. Create a comprehensive environment configuration file that handles all system components.

Create a `.env` file in your project root directory with the following configuration:

```
 # Database Configuration
DB_HOST=localhost
DB_PORT=5432
DB_NAME=odk_mcp_system
DB_USER=odk_user
DB_PASSWORD=secure_password_here
DB_SSL_MODE=prefer

# Redis Configuration
REDIS_HOST=localhost
REDIS_PORT=6379
REDIS_PASSWORD=redis_password_here
REDIS_DB=0

# Application Configuration
NODE_ENV=development
FLASK_ENV=development
SECRET_KEY=your_secret_key_here
JWT_SECRET=jwt_secret_key_here

# API Configuration
API_BASE_URL=http://localhost:5000
FORM_MANAGEMENT_PORT=5001
DATA_COLLECTION_PORT=5002
DATA_AGGREGATION_PORT=5003
```

```
# Frontend Configuration
REACT_APP_API_URL=http://localhost:5000
REACT_APP_ENVIRONMENT=development

# Mobile App Configuration
EXPO_PUBLIC_API_URL=http://localhost:5000
EXPO_PUBLIC_ENVIRONMENT=development

# Admin System Configuration
ADMIN_SECRET_KEY=admin_secret_key_here
ADMIN_PORT=5555

# AI Services Configuration
OPENAI_API_KEY=your_openai_key_here
ANTHROPIC_API_KEY=your_anthropic_key_here

# Payment Configuration (Development)
STRIPE_PUBLISHABLE_KEY=pk_test_your_stripe_key
STRIPE_SECRET_KEY=sk_test_your_stripe_secret
RAZORPAY_KEY_ID=your_razorpay_key
RAZORPAY_KEY_SECRET=your_razorpay_secret

# Email Configuration
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your_email@gmail.com
SMTP_PASSWORD=your_app_password

# File Storage Configuration
UPLOAD_FOLDER=./uploads
MAX_CONTENT_LENGTH=16777216  # 16MB
ALLOWED_EXTENSIONS=jpg,jpeg,png,gif,pdf,doc,docx,xls,xlsx

# Security Configuration
BCRYPT_LOG_ROUNDS=12
SESSION_TIMEOUT=3600  # 1 hour
MAX_LOGIN_ATTEMPTS=5
ACCOUNT_LOCKOUT_DURATION=1800  # 30 minutes

# Monitoring Configuration
LOG_LEVEL=INFO
LOG_FILE=./logs/application.log
METRICS_ENABLED=true
```

Development Tools Installation

Visual Studio Code Extensions:

Install essential extensions for optimal development experience:

```
# Install VS Code extensions via command line
code --install-extension ms-python.python
code --install-extension ms-vscode.vscode-typescript-next
code --install-extension bradlc.vscode-tailwindcss
code --install-extension ms-vscode.vscode-json
code --install-extension redhat.vscode-yaml
code --install-extension ms-python.flake8
code --install-extension ms-python.black-formatter
code --install-extension esbenp.prettier-vscode
```

Python Virtual Environment Setup:

Create isolated Python environments for clean dependency management:

```
# Create virtual environment
python3.11 -m venv venv

# Activate virtual environment (Linux/macOS)
source venv/bin/activate

# Activate virtual environment (Windows)
venv\Scripts\activate

# Upgrade pip and install essential packages
pip install --upgrade pip setuptools wheel

# Install project dependencies
pip install -r requirements.txt
```

Node.js Package Management:

Configure npm for optimal package installation and security:

```
# Set npm registry and configure security settings
npm config set registry https://registry.npmjs.org/
npm config set audit-level moderate

# Install global development tools
npm install -g @expo/cli react-native-cli nodemon

# Verify installations
node --version
npm --version
expo --version
```

Database Configuration

PostgreSQL Installation and Setup

PostgreSQL serves as the primary database system for the Enhanced ODK MCP System, providing robust data storage, advanced querying capabilities, and enterprise-grade security features. The setup process involves installing PostgreSQL, creating the necessary databases and users, and configuring security settings appropriate for development environments.

PostgreSQL Installation Verification:

After installing PostgreSQL through your system's package manager, verify the installation and start the database service:

```
 # Check PostgreSQL version
psql --version

# Start PostgreSQL service (Linux/macOS)
sudo systemctl start postgresql
sudo systemctl enable postgresql

# Start PostgreSQL service (macOS with Homebrew)
brew services start postgresql@15

# Start PostgreSQL service (Windows)
net start postgresql-x64-15
```

Database and User Creation:

Connect to PostgreSQL as the superuser and create the necessary database and user accounts with appropriate permissions:

```
 -- Connect to PostgreSQL as superuser
sudo -u postgres psql

-- Create main database
CREATE DATABASE odk_mcp_system;

-- Create application user with limited privileges
CREATE USER odk_user WITH PASSWORD 'secure_password_here';

-- Grant necessary permissions
GRANT CONNECT ON DATABASE odk_mcp_system TO odk_user;
GRANT USAGE ON SCHEMA public TO odk_user;
GRANT CREATE ON SCHEMA public TO odk_user;
```

```
-- Create additional databases for testing
CREATE DATABASE odk_mcp_system_test;
GRANT ALL PRIVILEGES ON DATABASE odk_mcp_system_test TO
odk_user;

-- Enable required extensions
\c odk_mcp_system
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION IF NOT EXISTS "pgcrypto";
CREATE EXTENSION IF NOT EXISTS "pg_stat_statements";

-- Exit PostgreSQL
\q
```

Database Schema Initialization:

The system requires specific database schemas for different components. Execute the schema creation script to set up all necessary tables, indexes, and constraints:

```
 # Navigate to project directory
cd /path/to/enhanced-odk-mcp-system

# Run database initialization script
python scripts/production_postgresql_setup.py

# Verify database structure
psql -h localhost -U odk_user -d odk_mcp_system -c "\dt"
```

Redis Configuration

Redis provides caching, session management, and real-time data processing capabilities. Configure Redis for development use with appropriate security settings and performance optimizations.

Redis Installation Verification:

```
 # Check Redis version
redis-server --version

# Start Redis server
redis-server

# Test Redis connection
redis-cli ping
# Expected response: PONG
```

Redis Configuration File:

Create a custom Redis configuration file for development use:

```
# Create Redis configuration directory
sudo mkdir -p /etc/redis

# Create development configuration file
sudo tee /etc/redis/redis-dev.conf << EOF
# Basic configuration
port 6379
bind 127.0.0.1
protected-mode yes

# Memory management
maxmemory 256mb
maxmemory-policy allkeys-lru

# Persistence
save 900 1
save 300 10
save 60 10000

# Security
requirepass redis_password_here

# Logging
loglevel notice
logfile /var/log/redis/redis-server.log

# Database
databases 16
EOF

# Start Redis with custom configuration
redis-server /etc/redis/redis-dev.conf
```

Database Migration and Seeding

Running Database Migrations:

The system includes comprehensive migration scripts that create all necessary tables, indexes, and initial data:

```
# Activate Python virtual environment
source venv/bin/activate

# Run database migrations
```

```
python scripts/run_migrations.py

# Verify migration status
python scripts/check_migration_status.py
```

Seeding Development Data:

Populate the database with sample data for development and testing:

```
# Run data seeding script
python scripts/seed_development_data.py

# Verify seeded data
psql -h localhost -U odk_user -d odk_mcp_system -c "SELECT
COUNT(*) FROM users;"
psql -h localhost -U odk_user -d odk_mcp_system -c "SELECT
COUNT(*) FROM organizations;"
```

Backend Services Setup

MCP Services Architecture

The Enhanced ODK MCP System utilizes a microservices architecture with three primary Model Context Protocol (MCP) servers handling different aspects of data management. Each service operates independently while maintaining seamless communication through well-defined APIs and shared database resources.

Service Overview:

1. **Form Management Service (Port 5001):** Handles form creation, validation, distribution, and version control
2. **Data Collection Service (Port 5002):** Manages data submission, validation, and initial processing
3. **Data Aggregation Service (Port 5003):** Provides analytics, reporting, and cross-project analysis capabilities

Starting Individual Services:

Each MCP service can be started independently for development and testing purposes. Navigate to the respective service directory and execute the startup commands:

```
# Start Form Management Service
cd mcps/form_management
```

```
source ../../venv/bin/activate
export FLASK_ENV=development
export FLASK_APP=src/main.py
flask run --host=0.0.0.0 --port=5001

# In a new terminal, start Data Collection Service
cd mcps/data_collection
source ../../venv/bin/activate
export FLASK_ENV=development
export FLASK_APP=src/main.py
flask run --host=0.0.0.0 --port=5002

# In another terminal, start Data Aggregation Service
cd mcps/data_aggregation
source ../../venv/bin/activate
export FLASK_ENV=development
export FLASK_APP=src/main.py
flask run --host=0.0.0.0 --port=5003
```

Service Health Verification:

After starting each service, verify their operational status using the health check endpoints:

```
 # Check Form Management Service
curl http://localhost:5001/health

# Check Data Collection Service
curl http://localhost:5002/health

# Check Data Aggregation Service
curl http://localhost:5003/health

# Expected response for each service:
# {"status": "healthy", "service": "service_name", "timestamp":
"2024-12-XX..."}
```

AI Services Configuration

The system includes comprehensive AI-powered features for data analysis, anomaly detection, and intelligent recommendations. These services require proper configuration and API key setup for optimal functionality.

AI Services Initialization:

```
 # Start AI Analytics Service
cd services
```

```
source ../venv/bin/activate
python analysis_templates.py &

# Start Virtual Assistant Service
python ../ai_modules/virtual_assistant.py &

# Start Anomaly Detection Service
python ../ai_modules/anomaly_detection/detector.py &
```

API Key Configuration:

Configure API keys for external AI services in your environment file:

```
 # OpenAI Configuration (for GPT-based features)
export OPENAI_API_KEY="your_openai_api_key_here"

# Anthropic Configuration (for Claude-based features)
export ANTHROPIC_API_KEY="your_anthropic_api_key_here"

# Verify API connectivity
python -c "
import openai
openai.api_key = 'your_openai_api_key_here'
try:
    response = openai.Model.list()
    print('OpenAI API connection successful')
except Exception as e:
    print(f'OpenAI API connection failed: {e}')
"
```

Subscription and Payment Services

The subscription system handles user billing, feature access control, and payment processing through multiple payment gateways.

Payment Gateway Configuration:

```
 # Start Subscription Service
cd services
source ../venv/bin/activate
python subscription_system.py &

# Configure Stripe (Development Mode)
export
STRIPE_PUBLISHABLE_KEY="pk_test_your_stripe_publishable_key"
export STRIPE_SECRET_KEY="sk_test_your_stripe_secret_key"

# Configure Razorpay (Development Mode)
```

```
export RAZORPAY_KEY_ID="your_razorpay_key_id"
export RAZORPAY_KEY_SECRET="your_razorpay_key_secret"

# Test payment gateway connectivity
python -c "
import stripe
stripe.api_key = 'sk_test_your_stripe_secret_key'
try:
    stripe.Account.retrieve()
    print('Stripe API connection successful')
except Exception as e:
    print(f'Stripe API connection failed: {e}')
"
```

Developer Admin System

The developer admin system provides comprehensive administrative controls and system monitoring capabilities exclusively for system developers.

Starting Admin System:

```
 # Start Developer Admin System
cd services
source ../venv/bin/activate
python developer_admin.py

# The admin system will start on port 5555
# Access URL: http://localhost:5555/admin/dashboard
# Default credentials:
# Username: developer_admin
# Password: DevAdmin@2024!
```

Admin System Features:

The admin system provides the following capabilities:

- **System Monitoring:** Real-time CPU, memory, and disk usage statistics
- **Database Administration:** Connection monitoring, table statistics, and backup creation
- **User Management:** Account suspension, activation, and activity monitoring
- **Security Audit:** Comprehensive audit log access and security event tracking
- **Maintenance Mode:** System-wide maintenance toggle with custom messaging

Service Orchestration

For development convenience, create a service orchestration script that starts all backend services simultaneously:

```bash
 #!/bin/bash
# File: scripts/start_all_services.sh

echo "Starting Enhanced ODK MCP System Services..."

# Start PostgreSQL and Redis if not running
sudo systemctl start postgresql redis-server

# Start MCP Services
echo "Starting Form Management Service..."
cd mcps/form_management && source ../../venv/bin/activate &&
flask run --host=0.0.0.0 --port=5001 &

echo "Starting Data Collection Service..."
cd mcps/data_collection && source ../../venv/bin/activate &&
flask run --host=0.0.0.0 --port=5002 &

echo "Starting Data Aggregation Service..."
cd mcps/data_aggregation && source ../../venv/bin/activate &&
flask run --host=0.0.0.0 --port=5003 &

# Start AI Services
echo "Starting AI Services..."
cd services && source ../venv/bin/activate && python
analysis_templates.py &
cd ai_modules && python virtual_assistant.py &

# Start Subscription Service
echo "Starting Subscription Service..."
cd services && python subscription_system.py &

# Start Admin System
echo "Starting Developer Admin System..."
cd services && python developer_admin.py &

echo "All services started successfully!"
echo "Service URLs:"
echo "- Form Management: http://localhost:5001"
echo "- Data Collection: http://localhost:5002"
echo "- Data Aggregation: http://localhost:5003"
echo "- Admin Dashboard: http://localhost:5555"

# Wait for user input to stop services
read -p "Press Enter to stop all services..."
```

```
# Stop all background processes
pkill -f "flask run"
pkill -f "python.*\.py"

echo "All services stopped."
```

Make the script executable and run it:

```
 chmod +x scripts/start_all_services.sh
./scripts/start_all_services.sh
```

Web Application Development

React Application Setup

The web application provides a modern, responsive interface for data collection, analysis, and system administration. Built with React 18 and modern JavaScript features, it offers an intuitive user experience across desktop and mobile browsers.

Installing Web Application Dependencies:

Navigate to the web application directory and install all required dependencies:

```
 # Navigate to web application directory
cd web-app

# Install dependencies using npm
npm install

# Install additional development dependencies
npm install --save-dev @types/react @types/react-dom eslint
prettier

# Verify installation
npm list --depth=0
```

Development Server Configuration:

The React development server provides hot reloading, error overlay, and debugging capabilities for efficient development:

```
 # Start development server
npm start
```

```
# The application will start on http://localhost:3000
# Hot reloading is enabled by default
```

Environment Configuration for Web App:

Create a `.env.local` file in the web-app directory for frontend-specific environment variables:

```
# API Configuration
REACT_APP_API_BASE_URL=http://localhost:5000
REACT_APP_FORM_MANAGEMENT_URL=http://localhost:5001
REACT_APP_DATA_COLLECTION_URL=http://localhost:5002
REACT_APP_DATA_AGGREGATION_URL=http://localhost:5003

# Authentication Configuration
REACT_APP_JWT_EXPIRY=3600
REACT_APP_SESSION_TIMEOUT=1800

# Feature Flags
REACT_APP_ENABLE_ANALYTICS=true
REACT_APP_ENABLE_SUBSCRIPTIONS=true
REACT_APP_ENABLE_AI_FEATURES=true

# Payment Configuration
REACT_APP_STRIPE_PUBLISHABLE_KEY=pk_test_your_stripe_key
REACT_APP_RAZORPAY_KEY_ID=your_razorpay_key

# Map Configuration
REACT_APP_MAPBOX_ACCESS_TOKEN=your_mapbox_token
REACT_APP_DEFAULT_MAP_CENTER_LAT=40.7128
REACT_APP_DEFAULT_MAP_CENTER_LNG=-74.0060

# Analytics Configuration
REACT_APP_GOOGLE_ANALYTICS_ID=GA_MEASUREMENT_ID
REACT_APP_ENABLE_ERROR_REPORTING=true
```

Component Development

Core Component Structure:

The application follows a modular component architecture with clear separation of concerns:

```
web-app/src/
├── components/
│   ├── ui/                # Reusable UI components
│   ├── forms/          # Form-related components
│   ├── charts/         # Data visualization components
│   ├── layout/         # Layout and navigation components
│   └── auth/           # Authentication components
├── pages/              # Page-level components
├── contexts/           # React context providers
├── hooks/              # Custom React hooks
├── services/           # API service functions
├── utils/              # Utility functions
└── styles/             # CSS and styling files
```

Key Component Examples:

Authentication Component:

```
// src/components/auth/LoginForm.jsx
import React, { useState, useContext } from 'react';
import { AuthContext } from '../../contexts/AuthContext';
import { NotificationContext } from '../../contexts/
NotificationContext';

const LoginForm = () => {
  const [credentials, setCredentials] = useState({ email: '',
password: '' });
  const [loading, setLoading] = useState(false);
  const { login } = useContext(AuthContext);
  const { showNotification } = useContext(NotificationContext);

  const handleSubmit = async (e) => {
    e.preventDefault();
    setLoading(true);

    try {
      await login(credentials.email, credentials.password);
      showNotification('Login successful', 'success');
    } catch (error) {
      showNotification('Login failed: ' + error.message,
'error');
    } finally {
      setLoading(false);
    }
  };

  return (
    <form onSubmit={handleSubmit} className="login-form">
      <div className="form-group">
        <label htmlFor="email">Email Address</label>
        <input
          type="email"
          id="email"
          value={credentials.email}
          onChange={(e) => setCredentials({...credentials,
email: e.target.value})}
          required
        />
      </div>

      <div className="form-group">
        <label htmlFor="password">Password</label>
        <input
          type="password"
          id="password"
          value={credentials.password}
          onChange={(e) => setCredentials({...credentials,
password: e.target.value})}
          required
        />
      </div>

      <button type="submit" disabled={loading} className="btn-
primary">
        {loading ? 'Signing In...' : 'Sign In'}
      </button>
    </form>
  );
};

export default LoginForm;
```

Data Visualization Component:

```
// src/components/charts/ImpactChart.jsx
import React, { useEffect, useState } from 'react';
import Plot from 'react-plotly.js';
import { dataAggregationService } from '../../services/
apiService';

const ImpactChart = ({ projectId, timeRange }) => {
  const [chartData, setChartData] = useState(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    const fetchChartData = async () => {
      try {
        const response = await
dataAggregationService.getImpactAnalysis(projectId, timeRange);
        setChartData(response.data);
      } catch (error) {
        console.error('Error fetching chart data:', error);
      } finally {
        setLoading(false);
      }
    };

    fetchChartData();
  }, [projectId, timeRange]);

  if (loading) return <div className="chart-loading">Loading
chart...</div>;
  if (!chartData) return <div className="chart-error">Unable to
load chart data</div>;

  return (
    <div className="impact-chart">
      <Plot
        data={[
          {
            x: chartData.dates,
            y: chartData.values,
            type: 'scatter',
            mode: 'lines+markers',
            marker: { color: '#2E86AB' },
            name: 'Impact Metrics'
          }
        ]}
        layout={{
          title: 'Program Impact Over Time',
          xaxis: { title: 'Date' },
          yaxis: { title: 'Impact Score' },
          responsive: true
        }}
        config={{ displayModeBar: false }}
        style={{ width: '100%', height: '400px' }}
      />
    </div>
  );
};

export default ImpactChart;
```

API Integration

Service Layer Implementation:

Create a comprehensive service layer for API communication:

```
 // src/services/apiService.js
const API_BASE_URL = process.env.REACT_APP_API_BASE_URL ||
'http://localhost:5000';
```

```
class ApiService {
  constructor(baseURL) {
    this.baseURL = baseURL;
    this.token = localStorage.getItem('authToken');
  }

  async request(endpoint, options = {}) {
    const url = `${this.baseURL}${endpoint}`;
    const config = {
      headers: {
        'Content-Type': 'application/json',
        ...(this.token && { Authorization: `Bearer ${this.token}
` }),
        ...options.headers,
      },
      ...options,
    };

    try {
      const response = await fetch(url, config);

      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}
`);
      }

      return await response.json();
    } catch (error) {
      console.error('API request failed:', error);
      throw error;
    }
  }

  // Authentication methods
  async login(email, password) {
    const response = await this.request('/auth/login', {
      method: 'POST',
      body: JSON.stringify({ email, password }),
    });

    if (response.token) {
      this.token = response.token;
      localStorage.setItem('authToken', response.token);
    }

    return response;
  }

  async logout() {
    await this.request('/auth/logout', { method: 'POST' });
    this.token = null;
    localStorage.removeItem('authToken');
  }

  // Form management methods
  async getForms() {
    return this.request('/forms');
  }

  async createForm(formData) {
    return this.request('/forms', {
      method: 'POST',
      body: JSON.stringify(formData),
    });
  }

  async updateForm(formId, formData) {
    return this.request(`/forms/${formId}`, {
      method: 'PUT',
      body: JSON.stringify(formData),
    });
  }

  // Data collection methods
  async submitFormData(formId, submissionData) {
    return this.request('/submissions', {
      method: 'POST',
      body: JSON.stringify({ formId, data: submissionData }),
    });
  }

  async getSubmissions(formId, filters = {}) {
    const queryParams = new URLSearchParams(filters).toString();
    return this.request(`/submissions?formId=${formId}&$
{queryParams}`);
  }

  // Analytics methods
  async getAnalytics(projectId, analysisType) {
    return this.request(`/analytics/${projectId}?type=$
{analysisType}`);
  }

  async generateReport(projectId, reportConfig) {
    return this.request(`/reports/generate`, {
      method: 'POST',
      body: JSON.stringify({ projectId, config: reportConfig }),
    });
  }
}

// Create service instances
export const apiService = new ApiService(API_BASE_URL);
export const formManagementService = new
```

```
ApiService(process.env.REACT_APP_FORM_MANAGEMENT_URL);
export const dataCollectionService = new
ApiService(process.env.REACT_APP_DATA_COLLECTION_URL);
export const dataAggregationService = new
ApiService(process.env.REACT_APP_DATA_AGGREGATION_URL);
```

Testing and Quality Assurance

Unit Testing Setup:

Configure Jest and React Testing Library for comprehensive component testing:

```
 # Install testing dependencies
npm install --save-dev @testing-library/react @testing-library/
jest-dom @testing-library/user-event

# Create test configuration file
# jest.config.js
module.exports = {
  testEnvironment: 'jsdom',
  setupFilesAfterEnv: ['<rootDir>/src/setupTests.js'],
  moduleNameMapping: {
    '\\.(css|less|scss|sass)$': 'identity-obj-proxy',
  },
  collectCoverageFrom: [
    'src/**/*.{js,jsx}',
    '!src/index.js',
    '!src/reportWebVitals.js',
  ],
  coverageThreshold: {
    global: {
      branches: 80,
      functions: 80,
      lines: 80,
      statements: 80,
    },
  },
};
```

Example Component Test:

```
 // src/components/auth/__tests__/LoginForm.test.js
import React from 'react';
import { render, screen, fireEvent, waitFor } from '@testing-
library/react';
import userEvent from '@testing-library/user-event';
import { AuthContext } from '../../../contexts/AuthContext';
import { NotificationContext } from '../../../contexts/
```

```
NotificationContext';
import LoginForm from '../LoginForm';

const mockLogin = jest.fn();
const mockShowNotification = jest.fn();

const renderWithContext = (component) => {
  return render(
    <AuthContext.Provider value={{ login: mockLogin }}>
      <NotificationContext.Provider value={{ showNotification:
mockShowNotification }}>
        {component}
      </NotificationContext.Provider>
    </AuthContext.Provider>
  );
};

describe('LoginForm', () => {
  beforeEach(() => {
    jest.clearAllMocks();
  });

  test('renders login form with email and password fields', ()
=> {
    renderWithContext(<LoginForm />);

    expect(screen.getByLabelText(/email address/
i)).toBeInTheDocument();
    expect(screen.getByLabelText(/password/
i)).toBeInTheDocument();
    expect(screen.getByRole('button', { name: /sign in/
i })).toBeInTheDocument();
  });

  test('submits form with correct credentials', async () => {
    const user = userEvent.setup();
    mockLogin.mockResolvedValue({ success: true });

    renderWithContext(<LoginForm />);

    await user.type(screen.getByLabelText(/email address/i),
'test@example.com');
    await user.type(screen.getByLabelText(/password/i),
'password123');
    await user.click(screen.getByRole('button', { name: /sign
in/i }));

    await waitFor(() => {

expect(mockLogin).toHaveBeenCalledWith('test@example.com',
'password123');
      expect(mockShowNotification).toHaveBeenCalledWith('Login
successful', 'success');
    });
  });

  test('displays error message on login failure', async () => {
    const user = userEvent.setup();
    mockLogin.mockRejectedValue(new Error('Invalid
credentials'));

    renderWithContext(<LoginForm />);

    await user.type(screen.getByLabelText(/email address/i),
'test@example.com');
    await user.type(screen.getByLabelText(/password/i),
'wrongpassword');
    await user.click(screen.getByRole('button', { name: /sign
in/i }));

    await waitFor(() => {
      expect(mockShowNotification).toHaveBeenCalledWith(
        'Login failed: Invalid credentials',
        'error'
      );
    });
  });
});
```

Running Tests:

```
# Run all tests
npm test

# Run tests with coverage
npm test -- --coverage

# Run tests in watch mode
npm test -- --watch

# Run specific test file
npm test LoginForm.test.js
```

Mobile Application Development

React Native Environment Setup

The mobile application provides offline-first data collection capabilities with automatic synchronization, QR code scanning, and comprehensive form management. Built with React Native and Expo, it supports both iOS and Android platforms with native performance and user experience.

Expo CLI Installation and Configuration:

Expo provides a comprehensive development platform for React Native applications, offering streamlined development, testing, and deployment workflows:

```
# Install Expo CLI globally
npm install -g @expo/cli

# Verify Expo installation
expo --version

# Install EAS CLI for building and deployment
npm install -g @expo/eas-cli

# Login to Expo account (create account at expo.dev if needed)
expo login
```

Android Development Environment:

For Android development and testing, install Android Studio and configure the Android SDK:

```
# Download and install Android Studio from https://
developer.android.com/studio

# After installation, open Android Studio and install:
# - Android SDK Platform 33 (Android 13)
# - Android SDK Platform 34 (Android 14)
# - Android SDK Build-Tools 33.0.0 and 34.0.0
# - Android Emulator
# - Intel x86 Emulator Accelerator (HAXM installer)

# Set environment variables (add to ~/.bashrc or ~/.zshrc)
export ANDROID_HOME=$HOME/Android/Sdk
export PATH=$PATH:$ANDROID_HOME/emulator
export PATH=$PATH:$ANDROID_HOME/platform-tools
export PATH=$PATH:$ANDROID_HOME/tools/bin
```

```
# Verify Android SDK installation
adb version
emulator -list-avds
```

iOS Development Environment (macOS only):

For iOS development, install Xcode and iOS Simulator:

```
 # Install Xcode from Mac App Store
# After installation, install command line tools:
xcode-select --install

# Install iOS Simulator
# Open Xcode > Preferences > Components > Install iOS Simulators

# Install CocoaPods for iOS dependency management
sudo gem install cocoapods

# Verify installation
pod --version
```

Mobile App Project Setup

Initializing the Mobile Application:

Navigate to the mobile app directory and install dependencies:

```
 # Navigate to mobile app directory
cd mobile-app

# Install dependencies
npm install

# Install additional React Native packages
npm install @react-navigation/native @react-navigation/stack
@react-navigation/bottom-tabs
npm install react-native-screens react-native-safe-area-context
npm install @react-native-async-storage/async-storage
npm install react-native-camera react-native-qrcode-scanner
npm install react-native-maps react-native-geolocation-service
npm install react-native-document-picker react-native-fs
npm install @reduxjs/toolkit react-redux redux-persist

# Install Expo-specific packages
expo install expo-camera expo-barcode-scanner expo-location
expo install expo-document-picker expo-file-system expo-sqlite
expo install expo-notifications expo-updates expo-splash-screen
```

Mobile App Configuration:

Create the app configuration file with proper settings for development and production:

```
// app.json
{
  "expo": {
    "name": "ODK MCP Mobile",
    "slug": "odk-mcp-mobile",
    "version": "1.0.0",
    "orientation": "portrait",
    "icon": "./assets/icon.png",
    "userInterfaceStyle": "automatic",
    "splash": {
      "image": "./assets/splash.png",
      "resizeMode": "contain",
      "backgroundColor": "#2E86AB"
    },
    "assetBundlePatterns": [
      "**/*"
    ],
    "ios": {
      "supportsTablet": true,
      "bundleIdentifier": "com.odk.mcp.mobile",
      "buildNumber": "1.0.0",
      "infoPlist": {
        "NSCameraUsageDescription": "This app uses camera to
capture photos for form submissions and QR code scanning.",
        "NSLocationWhenInUseUsageDescription": "This app uses
location to tag form submissions with geographic coordinates.",
        "NSMicrophoneUsageDescription": "This app uses
microphone to record audio for form submissions."
      }
    },
    "android": {
      "adaptiveIcon": {
        "foregroundImage": "./assets/adaptive-icon.png",
        "backgroundColor": "#2E86AB"
      },
      "package": "com.odk.mcp.mobile",
      "versionCode": 1,
      "permissions": [
        "CAMERA",
        "RECORD_AUDIO",
        "ACCESS_FINE_LOCATION",
        "ACCESS_COARSE_LOCATION",
        "READ_EXTERNAL_STORAGE",
        "WRITE_EXTERNAL_STORAGE"
      ]
    },
    "web": {
      "favicon": "./assets/favicon.png"
    },
    "plugins": [
      "expo-camera",
      "expo-barcode-scanner",
      "expo-location",
      "expo-document-picker",
      "expo-notifications"
    ],
    "extra": {
      "apiUrl": "http://localhost:5000",
      "environment": "development"
    }
  }
}
```

Environment Configuration for Mobile App:

Create environment-specific configuration files:

```
// config/environment.js
const ENV = {
  development: {
    apiUrl: 'http://localhost:5000',
    formManagementUrl: 'http://localhost:5001',
    dataCollectionUrl: 'http://localhost:5002',
    dataAggregationUrl: 'http://localhost:5003',
    enableLogging: true,
    enableCrashReporting: false,
    offlineStorageLimit: 100, // MB
    syncInterval: 300000, // 5 minutes
  },
  staging: {
    apiUrl: 'https://staging-api.odk-mcp.com',
    formManagementUrl: 'https://staging-forms.odk-mcp.com',
    dataCollectionUrl: 'https://staging-data.odk-mcp.com',
    dataAggregationUrl: 'https://staging-analytics.odk-mcp.com',
    enableLogging: true,
    enableCrashReporting: true,
    offlineStorageLimit: 500, // MB
    syncInterval: 180000, // 3 minutes
  },
  production: {
    apiUrl: 'https://api.odk-mcp.com',
    formManagementUrl: 'https://forms.odk-mcp.com',
    dataCollectionUrl: 'https://data.odk-mcp.com',
    dataAggregationUrl: 'https://analytics.odk-mcp.com',
    enableLogging: false,
    enableCrashReporting: true,
    offlineStorageLimit: 1000, // MB
    syncInterval: 120000, // 2 minutes
  },
};

const getEnvironment = () => {
  const env = process.env.NODE_ENV || 'development';
  return ENV[env] || ENV.development;
};

export default getEnvironment();
```

Core Mobile App Features

Offline Data Management:

Implement comprehensive offline data storage and synchronization:

```
 // src/services/OfflineStorageService.js
import AsyncStorage from '@react-native-async-storage/async-
storage';
import * as SQLite from 'expo-sqlite';
import NetInfo from '@react-native-netinfo/netinfo';

class OfflineStorageService {
  constructor() {
    this.db = SQLite.openDatabase('odk_offline.db');
    this.initializeDatabase();
  }

  initializeDatabase() {
    this.db.transaction(tx => {
      // Create tables for offline storage
      tx.executeSql(`
        CREATE TABLE IF NOT EXISTS offline_submissions (
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          form_id TEXT NOT NULL,
          submission_data TEXT NOT NULL,
          created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
          synced BOOLEAN DEFAULT FALSE,
          sync_attempts INTEGER DEFAULT 0
        )
      `);

      tx.executeSql(`
        CREATE TABLE IF NOT EXISTS offline_forms (
          id TEXT PRIMARY KEY,
          form_data TEXT NOT NULL,
          version INTEGER NOT NULL,
          downloaded_at DATETIME DEFAULT CURRENT_TIMESTAMP
        )
      `);

      tx.executeSql(`
        CREATE TABLE IF NOT EXISTS offline_media (
          id INTEGER PRIMARY KEY AUTOINCREMENT,
          submission_id INTEGER,
          file_path TEXT NOT NULL,
          file_type TEXT NOT NULL,
          synced BOOLEAN DEFAULT FALSE,
          FOREIGN KEY (submission_id) REFERENCES
offline_submissions (id)
        )
      `);
    });
  }

  async saveOfflineSubmission(formId, submissionData) {
    return new Promise((resolve, reject) => {
      this.db.transaction(
        tx => {
          tx.executeSql(
            'INSERT INTO offline_submissions (form_id,
submission_data) VALUES (?, ?)',
            [formId, JSON.stringify(submissionData)],
            (_, result) => resolve(result.insertId),
            (_, error) => reject(error)
          );
        },
        error => reject(error)
      );
    });
  }

  async getUnsyncedSubmissions() {
    return new Promise((resolve, reject) => {
      this.db.transaction(tx => {
        tx.executeSql(
          'SELECT * FROM offline_submissions WHERE synced =
FALSE ORDER BY created_at ASC',
          [],
          (_, { rows }) => resolve(rows._array),
          (_, error) => reject(error)
        );
      });
    });
  }

  async markSubmissionAsSynced(submissionId) {
    return new Promise((resolve, reject) => {
      this.db.transaction(tx => {
        tx.executeSql(
'UPDATE offline_submissions SET synced = TRUE WHERE id = ?',
          [submissionId],
          (_, result) => resolve(result),
          (_, error) => reject(error)
        );
      });
    });
  }

  async syncWithServer() {
    const isConnected = await NetInfo.fetch().then(state =>
state.isConnected);

    if (!isConnected) {
      console.log('No internet connection, skipping sync');
      return;
    }

    try {
      const unsyncedSubmissions = await
this.getUnsyncedSubmissions();

      for (const submission of unsyncedSubmissions) {
        try {
          const response = await fetch(`$
{config.dataCollectionUrl}/submissions`, {
            method: 'POST',
            headers: {
              'Content-Type': 'application/json',
              'Authorization': `Bearer ${await
this.getAuthToken()}`,
            },
            body: submission.submission_data,
          });

          if (response.ok) {
            await this.markSubmissionAsSynced(submission.id);
            console.log(`Synced submission ${submission.id}`);
          } else {
            console.error(`Failed to sync submission $
{submission.id}`);
          }
        } catch (error) {
          console.error(`Error syncing submission $
{submission.id}:`, error);
        }
      }
    } catch (error) {
      console.error('Error during sync:', error);
    }
  }
```

```
    async getAuthToken() {
      return await AsyncStorage.getItem('authToken');
    }
  }

  export default new OfflineStorageService();
```

QR Code Scanner Implementation:

```
// src/components/QRCodeScanner.js
import React, { useState, useEffect } from 'react';
import { View, Text, StyleSheet, Alert } from 'react-native';
import { BarCodeScanner } from 'expo-barcode-scanner';
import { Camera } from 'expo-camera';

const QRCodeScanner = ({ onScanSuccess, onScanError }) => {
  const [hasPermission, setHasPermission] = useState(null);
  const [scanned, setScanned] = useState(false);

  useEffect(() => {
    (async () => {
      const { status } = await
BarCodeScanner.requestPermissionsAsync();
      setHasPermission(status === 'granted');
    })();
  }, []);

  const handleBarCodeScanned = ({ type, data }) => {
    setScanned(true);

    try {
      // Parse QR code data
      const qrData = JSON.parse(data);

      if (qrData.type === 'form_access') {
        onScanSuccess({
          formId: qrData.formId,
          accessToken: qrData.accessToken,
          organizationId: qrData.organizationId,
        });
      } else if (qrData.type === 'project_access') {
        onScanSuccess({
          projectId: qrData.projectId,
          accessLevel: qrData.accessLevel,
          organizationId: qrData.organizationId,
        });
      } else {
        onScanError('Invalid QR code format');
      }
    } catch (error) {
      onScanError('Unable to parse QR code data');
    }
  };

  if (hasPermission === null) {
    return <Text>Requesting camera permission...</Text>;
  }

  if (hasPermission === false) {
    return <Text>No access to camera</Text>;
  }

  return (
    <View style={styles.container}>
      <BarCodeScanner
        onBarCodeScanned={scanned ? undefined :
handleBarCodeScanned}
        style={StyleSheet.absoluteFillObject}
      />

      <View style={styles.overlay}>
        <View style={styles.scanArea} />
        <Text style={styles.instructions}>
          Position QR code within the frame to scan
        </Text>
      </View>

      {scanned && (
        <View style={styles.resetContainer}>
          <Text
            style={styles.resetButton}
            onPress={() => setScanned(false)}
          >
            Tap to scan again
          </Text>
        </View>
      )}
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    flexDirection: 'column',
    justifyContent: 'center',
  },
  overlay: {
    position: 'absolute',
    top: 0,
    left: 0,
    right: 0,
    bottom: 0,
    justifyContent: 'center',
    alignItems: 'center',
  },
  scanArea: {
    width: 250,
    height: 250,
    borderWidth: 2,
    borderColor: '#2E86AB',
    borderRadius: 10,
    backgroundColor: 'transparent',
  },
  instructions: {
    marginTop: 20,
    fontSize: 16,
    color: 'white',
    textAlign: 'center',
    backgroundColor: 'rgba(0,0,0,0.5)',
    padding: 10,
    borderRadius: 5,
  },
  resetContainer: {
    position: 'absolute',
    bottom: 50,
    left: 0,
    right: 0,
    alignItems: 'center',
  },
  resetButton: {
    fontSize: 18,
    color: '#2E86AB',
    backgroundColor: 'white',
    padding: 15,
    borderRadius: 25,
    textAlign: 'center',
  },
});

export default QRCodeScanner;
```

Form Rendering Engine:

```
 // src/components/FormRenderer.js
import React, { useState, useEffect } from 'react';
import { View, Text, TextInput, TouchableOpacity, ScrollView,
StyleSheet } from 'react-native';
import { Picker } from '@react-native-picker/picker';
import DateTimePicker from '@react-native-community/
datetimepicker';
import * as ImagePicker from 'expo-image-picker';
import * as Location from 'expo-location';

const FormRenderer = ({ formDefinition, onSubmit, initialData =
{} }) => {
  const [formData, setFormData] = useState(initialData);
  const [errors, setErrors] = useState({});
  const [currentLocation, setCurrentLocation] = useState(null);

  useEffect(() => {
    getCurrentLocation();
  }, []);

  const getCurrentLocation = async () => {
    try {
      const { status } = await
Location.requestForegroundPermissionsAsync();
      if (status === 'granted') {
        const location = await
Location.getCurrentPositionAsync({});
        setCurrentLocation({
          latitude: location.coords.latitude,
          longitude: location.coords.longitude,
        });
      }
    } catch (error) {
      console.error('Error getting location:', error);
    }
  };

  const updateFormData = (fieldName, value) => {
    setFormData(prev => ({
      ...prev,
      [fieldName]: value,
    }));

    // Clear error when user starts typing
    if (errors[fieldName]) {
      setErrors(prev => ({
        ...prev,
        [fieldName]: null,
      }));
    }
  };

  const validateForm = () => {
    const newErrors = {};

    formDefinition.fields.forEach(field => {
      if (field.required && !formData[field.name]) {
        newErrors[field.name] = `${field.label} is required`;
      }
      if (field.type === 'email' && formData[field.name]) {
        const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        if (!emailRegex.test(formData[field.name])) {
          newErrors[field.name] = 'Please enter a valid email
address';
        }
      }

      if (field.type === 'number' && formData[field.name]) {
        if (isNaN(formData[field.name])) {
          newErrors[field.name] = 'Please enter a valid number';
        }
      }
    });

    setErrors(newErrors);
    return Object.keys(newErrors).length === 0;
  };

  const handleSubmit = () => {
    if (validateForm()) {
      const submissionData = {
        ...formData,
        _metadata: {
          submittedAt: new Date().toISOString(),
          location: currentLocation,
          deviceInfo: {
            platform: Platform.OS,
            version: Platform.Version,
          },
        },
      };

      onSubmit(submissionData);
    }
  };

  const renderField = (field) => {
    switch (field.type) {
      case 'text':
      case 'email':
        return (
          <View key={field.name} style={styles.fieldContainer}>
            <Text style={styles.fieldLabel}>
              {field.label}
              {field.required && <Text style={styles.required}>
*</Text>}
            </Text>
            <TextInput
              style={[styles.textInput, errors[field.name] &&
styles.errorInput]}
              value={formData[field.name] || ''}
              onChangeText={(value) =>
updateFormData(field.name, value)}
              placeholder={field.placeholder}
              keyboardType={field.type === 'email' ? 'email-
address' : 'default'}
            />
            {errors[field.name] && (
              <Text style={styles.errorText}
>{errors[field.name]}</Text>
            )}
          </View>
        );

      case 'number':
        return (
          <View key={field.name} style={styles.fieldContainer}>
            <Text style={styles.fieldLabel}>
              {field.label}
              {field.required && <Text style={styles.required}>
*</Text>}
            </Text>
            <TextInput
              style={[styles.textInput, errors[field.name] &&
styles.errorInput]}
              value={formData[field.name]?.toString() || ''}
              onChangeText={(value) =>
updateFormData(field.name, parseFloat(value) || '')}
              placeholder={field.placeholder}
              keyboardType="numeric"
            />
            {errors[field.name] && (
              <Text style={styles.errorText}
>{errors[field.name]}</Text>
            )}
          </View>
        );

      case 'select':
        return (
          <View key={field.name} style={styles.fieldContainer}>
            <Text style={styles.fieldLabel}>
              {field.label}
              {field.required && <Text style={styles.required}>
*</Text>}
            </Text>
            <View style={styles.pickerContainer}>
              <Picker
                selectedValue={formData[field.name] || ''}
                onValueChange={(value) =>
updateFormData(field.name, value)}
                style={styles.picker}
              >
```

```
                <Picker.Item label="Select an option..."
value="" />
                {field.options.map(option => (
                  <Picker.Item
                    key={option.value}
                    label={option.label}
                    value={option.value}
                  />
                ))}
              </Picker>
            </View>
            {errors[field.name] && (
              <Text style={styles.errorText}
>{errors[field.name]}</Text>
            )}
          </View>
        );

      case 'textarea':
        return (
          <View key={field.name} style={styles.fieldContainer}>
            <Text style={styles.fieldLabel}>
              {field.label}
              {field.required && <Text style={styles.required}>
*</Text>}
            </Text>
            <TextInput
              style={[styles.textArea, errors[field.name] &&
styles.errorInput]}
              value={formData[field.name] || ''}
              onChangeText={(value) =>
updateFormData(field.name, value)}
              placeholder={field.placeholder}
              multiline
              numberOfLines={4}
            />
            {errors[field.name] && (
              <Text style={styles.errorText}
>{errors[field.name]}</Text>
            )}
          </View>
        );

      default:
        return null;
    }
  };

  return (
    <ScrollView style={styles.container}>
      <Text style={styles.formTitle}>{formDefinition.title}</
Text>
```

```
      {formDefinition.description && (
        <Text style={styles.formDescription}
>{formDefinition.description}</Text>
      )}

      {formDefinition.fields.map(renderField)}

      <TouchableOpacity style={styles.submitButton}
onPress={handleSubmit}>
        <Text style={styles.submitButtonText}>Submit Form</Text>
      </TouchableOpacity>
    </ScrollView>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    padding: 20,
    backgroundColor: '#f5f5f5',
  },
  formTitle: {
    fontSize: 24,
    fontWeight: 'bold',
    marginBottom: 10,
    color: '#2E86AB',
  },
  formDescription: {
    fontSize: 16,
    marginBottom: 20,
    color: '#666',
  },
  fieldContainer: {
    marginBottom: 20,
  },
  fieldLabel: {
    fontSize: 16,
    fontWeight: '600',
    marginBottom: 5,
    color: '#333',
  },
  required: {
    color: '#e74c3c',
  },
  textInput: {
    borderWidth: 1,
    borderColor: '#ddd',
    borderRadius: 8,
    padding: 12,
    fontSize: 16,
    backgroundColor: 'white',
  },
  textArea: {
    borderWidth: 1,
    borderColor: '#ddd',
    borderRadius: 8,
    padding: 12,
    fontSize: 16,
    backgroundColor: 'white',
    minHeight: 100,
    textAlignVertical: 'top',
  },
  pickerContainer: {
    borderWidth: 1,
    borderColor: '#ddd',
    borderRadius: 8,
    backgroundColor: 'white',
  },
  picker: {
    height: 50,
  },
  errorInput: {
    borderColor: '#e74c3c',
  },
  errorText: {
    color: '#e74c3c',
    fontSize: 14,
    marginTop: 5,
  },
  submitButton: {
    backgroundColor: '#2E86AB',
    padding: 15,
    borderRadius: 8,
    alignItems: 'center',
    marginTop: 20,
    marginBottom: 40,
  },
  submitButtonText: {
    color: 'white',
    fontSize: 18,
    fontWeight: 'bold',
  },
});

export default FormRenderer;
```

Mobile App Testing

Running the Mobile App:

```
# Start the development server
cd mobile-app
```

```
expo start

# Options for testing:
# 1. Scan QR code with Expo Go app on physical device
# 2. Press 'a' to open Android emulator
# 3. Press 'i' to open iOS simulator (macOS only)
# 4. Press 'w' to open in web browser

# For Android emulator testing:
expo start --android

# For iOS simulator testing (macOS only):
expo start --ios

# For web testing:
expo start --web
```

Device Testing:

```
 # Install Expo Go app on your mobile device
# Available on Google Play Store and Apple App Store

# Connect to the same WiFi network as your development machine
# Scan the QR code displayed in terminal or browser

# For testing on physical device with USB debugging:
adb devices  # Verify device connection
expo start --localhost  # Use localhost tunnel
```

Building for Production:

```
 # Configure EAS Build
eas build:configure

# Build for Android
eas build --platform android

# Build for iOS (requires Apple Developer account)
eas build --platform ios

# Build for both platforms
eas build --platform all
```

Testing and Debugging

Comprehensive Testing Strategy

The Enhanced ODK MCP System employs a multi-layered testing approach ensuring reliability, performance, and user experience across all components. The testing strategy encompasses unit tests, integration tests, end-to-end tests, and performance testing.

Backend Testing Setup:

```
# Install testing dependencies
pip install pytest pytest-cov pytest-mock pytest-asyncio
pip install factory-boy faker responses

# Create pytest configuration
# pytest.ini
[tool:pytest]
testpaths = tests
python_files = test_*.py
python_classes = Test*
python_functions = test_*
addopts =
    --cov=src
    --cov-report=html
    --cov-report=term-missing
    --cov-fail-under=80
    -v
markers =
    unit: Unit tests
    integration: Integration tests
    e2e: End-to-end tests
    slow: Slow running tests
```

Example Backend Test:

```
# tests/test_form_management.py
import pytest
from unittest.mock import Mock, patch
from src.services.form_management import FormManagementService
from src.models.form import Form

class TestFormManagementService:
    @pytest.fixture
    def form_service(self):
        return FormManagementService()

    @pytest.fixture
    def sample_form_data(self):
        return {
            "title": "Health Survey",
            "description": "Community health assessment form",
            "fields": [
                {
                    "name": "age",
                    "type": "number",
                    "label": "Age",
                    "required": True
                },
                {
                    "name": "symptoms",
                    "type": "select",
                    "label": "Symptoms",
                    "options": ["fever", "cough", "fatigue"]
                }
            ]
        }

    def test_create_form_success(self, form_service,
sample_form_data):
        """Test successful form creation"""
        with patch.object(form_service.db, 'create_form') as
mock_create:
            mock_create.return_value = Form(id="form_123",
**sample_form_data)

            result = form_service.create_form(sample_form_data)

            assert result.id == "form_123"
            assert result.title == "Health Survey"

mock_create.assert_called_once_with(sample_form_data)

    def test_validate_form_structure(self, form_service,
sample_form_data):
        """Test form structure validation"""
        # Test valid form
        is_valid, errors =
form_service.validate_form_structure(sample_form_data)
        assert is_valid is True
        assert len(errors) == 0

        # Test invalid form (missing required fields)
        invalid_form = {"title": "Invalid Form"}
        is_valid, errors =
form_service.validate_form_structure(invalid_form)
        assert is_valid is False
        assert "fields" in errors

    @pytest.mark.integration
    def test_form_submission_workflow(self, form_service,
sample_form_data):
        """Test complete form submission workflow"""
        # Create form
        form = form_service.create_form(sample_form_data)

        # Submit data
        submission_data = {"age": 25, "symptoms": "fever"}
        submission = form_service.submit_form_data(form.id,
submission_data)

        assert submission.form_id == form.id
        assert submission.data["age"] == 25
        assert submission.status == "submitted"

# Run tests
# pytest tests/ -m unit
# pytest tests/ -m integration
# pytest tests/ --cov
```

Frontend Testing with Jest and React Testing Library:

```
 // web-app/src/components/__tests__/FormBuilder.test.js
import React from 'react';
import { render, screen, fireEvent, waitFor } from '@testing-
library/react';
import userEvent from '@testing-library/user-event';
import { Provider } from 'react-redux';
import { configureStore } from '@reduxjs/toolkit';
import FormBuilder from '../FormBuilder';
import { formBuilderSlice } from '../../store/formBuilderSlice';

const createTestStore = (initialState = {}) => {
  return configureStore({
    reducer: {
      formBuilder: formBuilderSlice.reducer,
    },
    preloadedState: {
      formBuilder: {
        fields: [],
        formTitle: '',
        formDescription: '',
        ...initialState,
      },
    },
  });
};

const renderWithStore = (component, initialState = {}) => {
  const store = createTestStore(initialState);
  return {
    ...render(
      <Provider store={store}>
        {component}
      </Provider>
    ),
    store,
  };
};

describe('FormBuilder', () => {
  test('renders form builder interface', () => {
    renderWithStore(<FormBuilder />);

    expect(screen.getByText(/form builder/
i)).toBeInTheDocument();
    expect(screen.getByPlaceholderText(/form title/
i)).toBeInTheDocument();
    expect(screen.getByText(/add field/i)).toBeInTheDocument();
  });

  test('adds new field to form', async () => {
    const user = userEvent.setup();
    renderWithStore(<FormBuilder />);

    // Click add field button
    await user.click(screen.getByText(/add field/i));

    // Select field type
    await user.selectOptions(screen.getByLabelText(/field type/
i), 'text');

    // Enter field details
    await user.type(screen.getByLabelText(/field label/i),
'Full Name');
    await user.type(screen.getByLabelText(/field name/i),
'full_name');

    // Save field
    await user.click(screen.getByText(/save field/i));

    // Verify field was added
    await waitFor(() => {
      expect(screen.getByText('Full Name')).toBeInTheDocument();
    });
  });

  test('validates form before saving', async () => {
    const user = userEvent.setup();
    renderWithStore(<FormBuilder />);

    // Try to save form without title
    await user.click(screen.getByText(/save form/i));
    // Check for validation error
    expect(screen.getByText(/form title is required/
i)).toBeInTheDocument();
  });
});
```

Mobile App Testing with Jest and React Native Testing Library:

```
// mobile-app/__tests__/FormRenderer.test.js
import React from 'react';
import { render, fireEvent, waitFor } from '@testing-library/
react-native';
import FormRenderer from '../src/components/FormRenderer';

const mockFormDefinition = {
  title: 'Test Form',
  description: 'A test form for unit testing',
  fields: [
    {
      name: 'name',
      type: 'text',
      label: 'Full Name',
      required: true,
      placeholder: 'Enter your full name',
    },
    {
      name: 'age',
      type: 'number',
      label: 'Age',
      required: true,
    },
    {
      name: 'gender',
      type: 'select',
      label: 'Gender',
      options: [
        { label: 'Male', value: 'male' },
        { label: 'Female', value: 'female' },
        { label: 'Other', value: 'other' },
      ],
    },
  ],
};

describe('FormRenderer', () => {
  const mockOnSubmit = jest.fn();

  beforeEach(() => {
    jest.clearAllMocks();
  });

  test('renders form with all fields', () => {
    const { getByText, getByPlaceholderText } = render(
      <FormRenderer
        formDefinition={mockFormDefinition}
        onSubmit={mockOnSubmit}
      />
    );

    expect(getByText('Test Form')).toBeTruthy();
    expect(getByText('A test form for unit
testing')).toBeTruthy();
    expect(getByText('Full Name')).toBeTruthy();
    expect(getByText('Age')).toBeTruthy();
    expect(getByText('Gender')).toBeTruthy();
    expect(getByPlaceholderText('Enter your full
name')).toBeTruthy();
  });

  test('validates required fields before submission', async ()
=> {
    const { getByText } = render(
      <FormRenderer
        formDefinition={mockFormDefinition}
        onSubmit={mockOnSubmit}
      />
    );

    // Try to submit without filling required fields
    fireEvent.press(getByText('Submit Form'));

    await waitFor(() => {
      expect(getByText('Full Name is required')).toBeTruthy();
      expect(getByText('Age is required')).toBeTruthy();
    });

    expect(mockOnSubmit).not.toHaveBeenCalled();
  });

  test('submits form with valid data', async () => {
    const { getByText, getByPlaceholderText } = render(
      <FormRenderer
        formDefinition={mockFormDefinition}
        onSubmit={mockOnSubmit}
      />
    );

    // Fill in required fields
    fireEvent.changeText(getByPlaceholderText('Enter your full
name'), 'John Doe');
    fireEvent.changeText(getByPlaceholderText('Age'), '25');
    // Submit form
    fireEvent.press(getByText('Submit Form'));

    await waitFor(() => {
      expect(mockOnSubmit).toHaveBeenCalledWith(
        expect.objectContaining({
          name: 'John Doe',
          age: '25',
          _metadata: expect.objectContaining({
            submittedAt: expect.any(String),
          }),
        })
      );
    });
  });
});
```

Performance Testing

Backend Performance Testing:

```
# tests/performance/test_api_performance.py
import pytest
import time
import concurrent.futures
import requests
from statistics import mean, median

class TestAPIPerformance:
    BASE_URL = "http://localhost:5000"

    def test_form_creation_performance(self):
        """Test form creation API performance"""
        def create_form():
            start_time = time.time()
            response = requests.post(
                f"{self.BASE_URL}/forms",
                json={
                    "title": f"Performance Test Form
{time.time()}",
                    "fields": [
                        {"name": "field1", "type": "text",
"label": "Field 1"},
                        {"name": "field2", "type": "number",
"label": "Field 2"},
                    ]
                }
            )
            end_time = time.time()
            return end_time - start_time, response.status_code

        # Test with concurrent requests
        with
concurrent.futures.ThreadPoolExecutor(max_workers=10) as
executor:
            futures = [executor.submit(create_form) for _ in
range(50)]
            results = [future.result() for future in futures]

        response_times = [result[0] for result in results]
        status_codes = [result[1] for result in results]

        # Performance assertions
        assert mean(response_times) < 1.0  # Average response
time < 1 second
        assert median(response_times) < 0.5  # Median response
time < 0.5 seconds
        assert max(response_times) < 3.0  # Max response time <
3 seconds
        assert all(code == 201 for code in status_codes)  # All
requests successful

    def test_database_query_performance(self):
        """Test database query performance"""
        from src.models.form import Form

        start_time = time.time()

        # Simulate complex query
        forms = Form.query.join(Submission).filter(
            Form.created_at >= '2024-01-01'
        ).limit(1000).all()

        end_time = time.time()
        query_time = end_time - start_time

        assert query_time < 2.0  # Query should complete in < 2
seconds
        assert len(forms) <= 1000  # Verify limit is respected
```

Frontend Performance Testing:

```
// web-app/src/utils/performanceMonitor.js
class PerformanceMonitor {
  constructor() {
    this.metrics = {};
  }

  startTimer(label) {
    this.metrics[label] = {
      startTime: performance.now(),
    };
  }

  endTimer(label) {
    if (this.metrics[label]) {
      this.metrics[label].endTime = performance.now();
      this.metrics[label].duration =
        this.metrics[label].endTime -
this.metrics[label].startTime;

      console.log(`${label}: $
{this.metrics[label].duration.toFixed(2)}ms`);
      return this.metrics[label].duration;
    }
  }

  measureComponentRender(Component) {
    return function MeasuredComponent(props) {
      const renderStart = performance.now();

      useEffect(() => {
        const renderEnd = performance.now();
        console.log(`${Component.name} render time: $
{(renderEnd - renderStart).toFixed(2)}ms`);
      });

      return <Component {...props} />;
    };
  }

  measureAPICall(apiFunction) {
    return async function(...args) {
      const start = performance.now();
      try {
        const result = await apiFunction(...args);
        const end = performance.now();
        console.log(`API call duration: ${(end -
start).toFixed(2)}ms`);
        return result;
      } catch (error) {
        const end = performance.now();
        console.log(`API call failed after: ${(end -
start).toFixed(2)}ms`);
        throw error;
      }
    };
  }
}

export default new PerformanceMonitor();
```

Debugging Tools and Techniques

Backend Debugging:

```
 # src/utils/debug_tools.py
import logging
import functools
import time
from flask import request, g

# Configure logging
logging.basicConfig(
    level=logging.DEBUG,
    format='%(asctime)s - %(name)s - %(levelname)s - %
(message)s',
    handlers=[
        logging.FileHandler('debug.log'),
        logging.StreamHandler()
    ]
)

def debug_api_calls(f):
    """Decorator to debug API calls"""
    @functools.wraps(f)
    def decorated_function(*args, **kwargs):
        start_time = time.time()

        logging.debug(f"API Call: {request.method}
{request.path}")
        logging.debug(f"Headers: {dict(request.headers)}")
        logging.debug(f"Args: {request.args}")
        logging.debug(f"JSON: {request.get_json()}")

        try:
            result = f(*args, **kwargs)
            end_time = time.time()
            logging.debug(f"API Call completed in {end_time -
start_time:.3f}s")
            return result
        except Exception as e:
            end_time = time.time()
            logging.error(f"API Call failed after {end_time -
start_time:.3f}s: {str(e)}")
            raise

    return decorated_function

def debug_database_queries():
    """Enable SQLAlchemy query debugging"""
    import sqlalchemy
```

```
logging.getLogger('sqlalchemy.engine').setLevel(logging.INFO)

# Usage in Flask routes
@app.route('/forms', methods=['POST'])
@debug_api_calls
def create_form():
    # Route implementation
    pass
```

Frontend Debugging:

```
// web-app/src/utils/debugTools.js
class DebugTools {
  constructor() {
    this.isDebugMode = process.env.NODE_ENV === 'development';
  }

  logAPICall(url, method, data, response) {
    if (this.isDebugMode) {
      console.group(`🌐 API Call: ${method} ${url}`);
      console.log('Request Data:', data);
      console.log('Response:', response);
      console.groupEnd();
    }
  }

  logStateChange(componentName, oldState, newState) {
    if (this.isDebugMode) {
      console.group(`🔄 State Change: ${componentName}`);
      console.log('Old State:', oldState);
      console.log('New State:', newState);
      console.groupEnd();
    }
  }

  logError(error, context) {
    console.group('❌ Error');
    console.error('Error:', error);
    console.log('Context:', context);
    console.log('Stack Trace:', error.stack);
    console.groupEnd();
  }

  measurePerformance(label, fn) {
    if (this.isDebugMode) {
      const start = performance.now();
      const result = fn();
      const end = performance.now();
      console.log(`⏱️ ${label}: ${(end - start).toFixed(2)}ms`);
      return result;
    }
    return fn();
  }
}

export default new DebugTools();
```

Troubleshooting Guide

Common Issues and Solutions

Database Connection Issues:

```
 # Issue: PostgreSQL connection refused
# Solution: Check if PostgreSQL is running
sudo systemctl status postgresql

# If not running, start it
sudo systemctl start postgresql

# Check if user exists and has correct permissions
sudo -u postgres psql -c "\du"

# Reset user password if needed
sudo -u postgres psql -c "ALTER USER odk_user PASSWORD
'new_password';"
```

Port Conflicts:

```
 # Issue: Port already in use
# Solution: Find and kill process using the port
lsof -ti:5001 | xargs kill -9

# Or use a different port
export FLASK_RUN_PORT=5004
flask run
```

Node.js Memory Issues:

```
 # Issue: JavaScript heap out of memory
# Solution: Increase Node.js memory limit
export NODE_OPTIONS="--max-old-space-size=4096"
npm start
```

Mobile App Build Issues:

```
# Issue: Android build fails
# Solution: Clean and rebuild
cd mobile-app
expo prebuild --clean
eas build --platform android --clear-cache

# Issue: iOS build fails (macOS)
# Solution: Update CocoaPods and clean
cd ios
pod deintegrate
pod install
cd ..
expo run:ios
```

Performance Optimization

Database Optimization:

```
-- Add indexes for frequently queried columns
CREATE INDEX idx_forms_organization_id ON
forms(organization_id);
CREATE INDEX idx_submissions_form_id ON submissions(form_id);
CREATE INDEX idx_submissions_created_at ON
submissions(created_at);

-- Analyze query performance
EXPLAIN ANALYZE SELECT * FROM forms WHERE organization_id =
'org_123';

-- Update table statistics
ANALYZE forms;
ANALYZE submissions;
```

Frontend Optimization:

```
// Implement code splitting
const LazyDashboard = React.lazy(() => import('./pages/
Dashboard'));
const LazyAnalytics = React.lazy(() => import('./pages/
Analytics'));

// Use React.memo for expensive components
const ExpensiveComponent = React.memo(({ data }) => {
  return <div>{/* Expensive rendering logic */}</div>;
});
```

```
// Implement virtual scrolling for large lists
import { FixedSizeList as List } from 'react-window';

const VirtualizedList = ({ items }) => (
  <List
    height={600}
    itemCount={items.length}
    itemSize={50}
    itemData={items}
  >
    {({ index, style, data }) => (
      <div style={style}>
        {data[index].name}
      </div>
    )}
  </List>
);
```

Production Deployment

Environment Preparation

Production Environment Variables:

```
 # Production .env file
NODE_ENV=production
FLASK_ENV=production

# Database Configuration
DB_HOST=your-production-db-host
DB_PORT=5432
DB_NAME=odk_mcp_production
DB_USER=odk_prod_user
DB_PASSWORD=secure_production_password
DB_SSL_MODE=require

# Redis Configuration
REDIS_HOST=your-production-redis-host
REDIS_PORT=6379
REDIS_PASSWORD=secure_redis_password

# Security Configuration
SECRET_KEY=your-super-secure-secret-key
JWT_SECRET=your-jwt-secret-key
BCRYPT_LOG_ROUNDS=12
```

```
# API Configuration
API_BASE_URL=https://api.yourdomain.com
ALLOWED_ORIGINS=https://yourdomain.com,https://
app.yourdomain.com

# Payment Configuration
STRIPE_PUBLISHABLE_KEY=pk_live_your_live_stripe_key
STRIPE_SECRET_KEY=sk_live_your_live_stripe_secret
RAZORPAY_KEY_ID=your_live_razorpay_key
RAZORPAY_KEY_SECRET=your_live_razorpay_secret

# Email Configuration
SMTP_HOST=your-smtp-host
SMTP_PORT=587
SMTP_USER=your-email@yourdomain.com
SMTP_PASSWORD=your-email-password

# Monitoring Configuration
SENTRY_DSN=your-sentry-dsn
LOG_LEVEL=WARNING
```

Docker Deployment:

```
 # Dockerfile for backend services
FROM python:3.11-slim

WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    postgresql-client \
    && rm -rf /var/lib/apt/lists/*

# Install Python dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt

# Copy application code
COPY . .

# Create non-root user
RUN useradd --create-home --shell /bin/bash app
USER app

# Expose port
EXPOSE 5000

# Start application
CMD ["gunicorn", "--bind", "0.0.0.0:5000", "--workers", "4",
"app:app"]
```

```yaml
# docker-compose.yml for complete system
version: '3.8'

services:
  postgres:
    image: postgres:15
    environment:
      POSTGRES_DB: odk_mcp_production
      POSTGRES_USER: odk_prod_user
      POSTGRES_PASSWORD: secure_production_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"

  redis:
    image: redis:7-alpine
    command: redis-server --requirepass secure_redis_password
    volumes:
      - redis_data:/data
    ports:
      - "6379:6379"

  form-management:
    build: ./mcps/form_management
    environment:
      - DB_HOST=postgres
      - REDIS_HOST=redis
    depends_on:
      - postgres
      - redis
    ports:
      - "5001:5000"

  data-collection:
    build: ./mcps/data_collection
    environment:
      - DB_HOST=postgres
      - REDIS_HOST=redis
    depends_on:
      - postgres
      - redis
    ports:
      - "5002:5000"

  data-aggregation:
    build: ./mcps/data_aggregation
    environment:
      - DB_HOST=postgres
      - REDIS_HOST=redis
    depends_on:
      - postgres
      - redis
    ports:
      - "5003:5000"

  web-app:
    build: ./web-app
    ports:
      - "3000:3000"
    environment:
      - REACT_APP_API_URL=https://api.yourdomain.com

volumes:
  postgres_data:
  redis_data:
```

Deployment Script:

```
#!/bin/bash
# deploy.sh

set -e

echo "Starting deployment..."

# Build and push Docker images
docker-compose build
docker-compose push

# Deploy to production server
ssh production-server << 'EOF'
  cd /opt/odk-mcp-system
  git pull origin main
  docker-compose down
  docker-compose pull
  docker-compose up -d

  # Run database migrations
  docker-compose exec data-aggregation python scripts/
run_migrations.py

  # Restart services
  docker-compose restart
EOF

echo "Deployment completed successfully!"
```

Monitoring and Maintenance

Health Check Endpoints:

```
# src/health_check.py
from flask import Blueprint, jsonify
import psutil
import redis
from sqlalchemy import text

health_bp = Blueprint('health', __name__)

@health_bp.route('/health')
def health_check():
    """Comprehensive health check"""
    status = {
        'status': 'healthy',
        'timestamp': datetime.utcnow().isoformat(),
        'checks': {}
    }

    # Database check
    try:
        db.session.execute(text('SELECT 1'))
        status['checks']['database'] = 'healthy'
    except Exception as e:
        status['checks']['database'] = f'unhealthy: {str(e)}'
        status['status'] = 'unhealthy'

    # Redis check
    try:
        r = redis.Redis(host=REDIS_HOST, port=REDIS_PORT,
password=REDIS_PASSWORD)
        r.ping()
        status['checks']['redis'] = 'healthy'
    except Exception as e:
        status['checks']['redis'] = f'unhealthy: {str(e)}'
        status['status'] = 'unhealthy'

    # System resources
    status['checks']['cpu_usage'] = f"{psutil.cpu_percent()}%"
    status['checks']['memory_usage'] =
f"{psutil.virtual_memory().percent}%"
    status['checks']['disk_usage'] =
f"{psutil.disk_usage('/').percent}%"

    return jsonify(status)

@health_bp.route('/metrics')
def metrics():
    """Prometheus-compatible metrics"""
    metrics_data = [
        f"cpu_usage_percent {psutil.cpu_percent()}",
        f"memory_usage_percent
{psutil.virtual_memory().percent}",
        f"disk_usage_percent {psutil.disk_usage('/').percent}",
    ]

    return '\n'.join(metrics_data), 200, {'Content-Type': 'text/
plain'}
```

This comprehensive local development setup guide provides everything needed to run the Enhanced ODK MCP System locally, including detailed instructions for both web and mobile applications, testing procedures, debugging tools, and production deployment strategies.
