🚀 COMPLETE MVP PROMPT - LINE BREAK DETECTION SYSTEM

📋 PROJECT OVERVIEW

Project Name: Kerala LT Line Break Detection System

Tech Stack: Node.js (TypeScript) + React.js (TypeScript) + Supabase + Vercel

Timeline: 3-4 weeks

Team: 3-5 developers

🎯 SYSTEM REQUIREMENTS SUMMARY

Core Features:

Real-time Line Break Detection - AI-powered fault detection

Live Dashboard - Monitor grid health and alerts

Alert System - SMS/Email notifications to field crews

Historical Analytics - Trend analysis and reporting

User Management - Role-based access (Admin, Operator, Field Crew)

Mobile App - React Native for field crew

📊 REQUIRED CREDENTIALS & CONFIGURATION

Before Starting - Information Needed:

bash

I NEED FROM YOU:

1. Supabase Project Details:

- Supabase Project URL: [YOUR_SUPABASE_URL]

- Supabase Anon Key: [YOUR_SUPABASE_ANON_KEY]

- Supabase Service Role Key: [YOUR_SUPABASE_SERVICE_KEY]

2. Email Service (for alerts):

- SMTP Host: [e.g., smtp.gmail.com]

- SMTP Port: [e.g., 587]

- Email: [YOUR_EMAIL]

- Email Password: [YOUR_APP_PASSWORD]

3. SMS Service (Twilio/MSG91):

- Account SID: [YOUR_TWILIO_SID]
- Auth Token: [YOUR_TWILIO_TOKEN]
- Phone Number: [YOUR_TWILIO_PHONE]

4. Vercel Deployment:

- Vercel Account: [YOUR_VERCEL_EMAIL]
- GitHub Repo: [YOUR_GITHUB_REPO_URL]

5. Optional (for production):

- OpenAI API Key: [for AI features]
- AWS S3 Credentials: [for large dataset storage]

🗄️ DATABASE SCHEMA (SUPABASE)

SQL Schema to Execute in Supabase:

sql

```
-- Enable UUID extension
CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

-- Users table
CREATE TABLE users (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  email VARCHAR(255) UNIQUE NOT NULL,
  password_hash TEXT NOT NULL,
  full_name VARCHAR(255) NOT NULL,
  role VARCHAR(50) NOT NULL CHECK (role IN ('admin', 'operator', 'field_crew')),
  phone VARCHAR(20),
  is_active BOOLEAN DEFAULT true,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  updated_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
```

```
);

-- Substations table
CREATE TABLE substations (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  name VARCHAR(255) NOT NULL,
  code VARCHAR(50) UNIQUE NOT NULL,
  location_lat DECIMAL(10, 8),
  location_lng DECIMAL(11, 8),
  address TEXT,
  voltage_level VARCHAR(50),
  capacity_mva DECIMAL(10, 2),
  is_active BOOLEAN DEFAULT true,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Feeders table
CREATE TABLE feeders (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  substation_id UUID REFERENCES substations(id) ON DELETE CASCADE,
  name VARCHAR(255) NOT NULL,
  code VARCHAR(50) UNIQUE NOT NULL,
  length_km DECIMAL(10, 2),
  conductor_type VARCHAR(100),
  line_impedance_real DECIMAL(10, 4),
  line_impedance_imag DECIMAL(10, 4),
  typical_load_kw DECIMAL(10, 2),
  num_consumers INTEGER,
  area_type VARCHAR(50) CHECK (area_type IN ('urban', 'rural', 'semi-urban')),
  is_active BOOLEAN DEFAULT true,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
```

```
);

-- Line break events table
CREATE TABLE line_break_events (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  feeder_id UUID REFERENCES feeders(id) ON DELETE CASCADE,
  detected_at TIMESTAMP WITH TIME ZONE NOT NULL,
  detection_method VARCHAR(50) DEFAULT 'ai_model',
  confidence_score DECIMAL(5, 4),
  estimated_location_km DECIMAL(10, 3),
  fault_type VARCHAR(50) DEFAULT 'LINE_BREAK',
  severity VARCHAR(50) CHECK (severity IN ('low', 'medium', 'high', 'critical')),
  status VARCHAR(50) DEFAULT 'detected' CHECK (status IN ('detected', 'acknowledged',
'crew_dispatched', 'resolved')),
  breaker_tripped BOOLEAN DEFAULT false,
  trip_time_ms INTEGER,
  resolved_at TIMESTAMP WITH TIME ZONE,
  resolution_notes TEXT,
  assigned_to UUID REFERENCES users(id),
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Waveform data table (for storing raw sensor data)
CREATE TABLE waveform_data (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  feeder_id UUID REFERENCES feeders(id) ON DELETE CASCADE,
  timestamp TIMESTAMP WITH TIME ZONE NOT NULL,
  current_r DECIMAL(10, 3)[],  -- Array of values
  current_y DECIMAL(10, 3)[],
  current_b DECIMAL(10, 3)[],
  voltage_r DECIMAL(10, 3)[],
```

```
  voltage_y DECIMAL(10, 3)[],
  voltage_b DECIMAL(10, 3)[],
  sampling_rate INTEGER DEFAULT 10000,
  duration_seconds DECIMAL(5, 2),
  label VARCHAR(50),
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Feature data table (extracted features for ML)
CREATE TABLE feature_data (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  waveform_id UUID REFERENCES waveform_data(id) ON DELETE CASCADE,
  features JSONB NOT NULL,  -- Store all features as JSON
  label VARCHAR(50),
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Alerts table
CREATE TABLE alerts (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  event_id UUID REFERENCES line_break_events(id) ON DELETE CASCADE,
  alert_type VARCHAR(50) CHECK (alert_type IN ('email', 'sms', 'push', 'dashboard')),
  recipient_id UUID REFERENCES users(id),
  recipient_contact VARCHAR(255),
  message TEXT,
  sent_at TIMESTAMP WITH TIME ZONE,
  status VARCHAR(50) DEFAULT 'pending' CHECK (status IN ('pending', 'sent', 'failed')),
  error_message TEXT,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);
```

```
-- System logs table
CREATE TABLE system_logs (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  level VARCHAR(20) CHECK (level IN ('info', 'warning', 'error', 'critical')),
  module VARCHAR(100),
  message TEXT,
  metadata JSONB,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Model performance metrics table
CREATE TABLE model_metrics (
  id UUID PRIMARY KEY DEFAULT uuid_generate_v4(),
  model_version VARCHAR(50),
  accuracy DECIMAL(5, 4),
  precision DECIMAL(5, 4),
  recall DECIMAL(5, 4),
  f1_score DECIMAL(5, 4),
  false_positive_rate DECIMAL(5, 4),
  evaluation_date TIMESTAMP WITH TIME ZONE DEFAULT NOW(),
  dataset_size INTEGER,
  notes TEXT,
  created_at TIMESTAMP WITH TIME ZONE DEFAULT NOW()
);

-- Create indexes for performance
CREATE INDEX idx_line_break_events_feeder ON line_break_events(feeder_id);
CREATE INDEX idx_line_break_events_detected_at ON line_break_events(detected_at DESC);
CREATE INDEX idx_line_break_events_status ON line_break_events(status);
CREATE INDEX idx_waveform_data_feeder ON waveform_data(feeder_id);
CREATE INDEX idx_waveform_data_timestamp ON waveform_data(timestamp DESC);
```

```
CREATE INDEX idx_alerts_event ON alerts(event_id);
CREATE INDEX idx_alerts_recipient ON alerts(recipient_id);

-- Enable Row Level Security (RLS)
ALTER TABLE users ENABLE ROW LEVEL SECURITY;
ALTER TABLE line_break_events ENABLE ROW LEVEL SECURITY;
ALTER TABLE alerts ENABLE ROW LEVEL SECURITY;

-- RLS Policies (basic example - expand as needed)
CREATE POLICY "Users can view their own data"
  ON users FOR SELECT
  USING (auth.uid() = id);

CREATE POLICY "Admins and operators can view all events"
  ON line_break_events FOR SELECT
  USING (
    EXISTS (
      SELECT 1 FROM users
      WHERE users.id = auth.uid()
      AND users.role IN ('admin', 'operator')
    )
  );
```

📁 PROJECT STRUCTURE

```
kerala-line-break-detection/
|
├── .env.example              # Template for environment variables
├── .env                      # Actual secrets (gitignored)
├── .cursorrules              # Cursor AI rules (JSON format)
├── .gitignore
├── package.json
```

```
├── tsconfig.json
├── vercel.json                # Vercel deployment config
│
├── backend/                   # Node.js + TypeScript API
│   ├── src/
│   │   ├── config/
│   │   │   ├── supabase.ts    # Supabase client
│   │   │   ├── env.ts         # Environment validation
│   │   │   └── database.ts    # DB connection pool
│   │   │
│   │   ├── models/            # Data models
│   │   │   ├── User.ts
│   │   │   ├── Substation.ts
│   │   │   ├── Feeder.ts
│   │   │   ├── LineBreakEvent.ts
│   │   │   └── Alert.ts
│   │   │
│   │   ├── controllers/       # Business logic
│   │   │   ├── authController.ts
│   │   │   ├── substationController.ts
│   │   │   ├── feederController.ts
│   │   │   ├── eventController.ts
│   │   │   ├── waveformController.ts
│   │   │   └── alertController.ts
│   │   │
│   │   ├── services/          # Core services
│   │   │   ├── mlService.ts          # ML model inference
│   │   │   ├── featureExtractor.ts   # Feature extraction
│   │   │   ├── alertService.ts       # Email/SMS alerts
│   │   │   ├── dataGenerator.ts      # Synthetic data
```

```
│   │   └── realTimeProcessor.ts   # Stream processing
│   │
│   ├── routes/            # API routes
│   │   ├── auth.routes.ts
│   │   ├── substation.routes.ts
│   │   ├── feeder.routes.ts
│   │   ├── event.routes.ts
│   │   ├── waveform.routes.ts
│   │   └── dashboard.routes.ts
│   │
│   ├── middleware/
│   │   ├── auth.middleware.ts     # JWT validation
│   │   ├── error.middleware.ts
│   │   ├── validation.middleware.ts
│   │   └── rateLimit.middleware.ts
│   │
│   ├── utils/
│   │   ├── logger.ts
│   │   ├── response.ts
│   │   └── validator.ts
│   │
│   ├── types/
│   │   ├── index.d.ts
│   │   └── api.types.ts
│   │
│   ├── ml/                # Machine Learning
│   │   ├── model.json      # Trained model
│   │   ├── scaler.json     # Feature scaler
│   │   └── inference.ts    # Prediction logic
│   │
```

```
|   |   └── server.ts          # Express app entry
|   |
|   ├── tests/
|   |   ├── unit/
|   |   └── integration/
|   |
|   └── package.json
|
├── frontend/                  # React + TypeScript
|   ├── public/
|   |   └── assets/
|   |
|   ├── src/
|   |   ├── api/               # API client
|   |   |   ├── client.ts      # Axios instance
|   |   |   ├── auth.api.ts
|   |   |   ├── substation.api.ts
|   |   |   ├── feeder.api.ts
|   |   |   ├── event.api.ts
|   |   |   └── dashboard.api.ts
|   |   |
|   |   ├── components/        # Reusable components
|   |   |   ├── common/
|   |   |   |   ├── Button.tsx
|   |   |   |   ├── Card.tsx
|   |   |   |   ├── Input.tsx
|   |   |   |   ├── Modal.tsx
|   |   |   |   └── Table.tsx
|   |   |   |
|   |   |   ├── layout/
```

```
|   |   |   |   ├── Header.tsx
|   |   |   |   ├── Sidebar.tsx
|   |   |   |   └── Footer.tsx
|   |   |   |
|   |   |   ├── dashboard/
|   |   |   |   ├── LiveMap.tsx
|   |   |   |   ├── AlertPanel.tsx
|   |   |   |   ├── StatsCard.tsx
|   |   |   |   ├── WaveformChart.tsx
|   |   |   |   └── EventTimeline.tsx
|   |   |   |
|   |   |   └── forms/
|   |   |       ├── LoginForm.tsx
|   |   |       ├── SubstationForm.tsx
|   |   |       └── FeederForm.tsx
|   |   |
|   |   ├── pages/            # Route pages
|   |   |   ├── LoginPage.tsx
|   |   |   ├── DashboardPage.tsx
|   |   |   ├── SubstationsPage.tsx
|   |   |   ├── FeedersPage.tsx
|   |   |   ├── EventsPage.tsx
|   |   |   ├── AnalyticsPage.tsx
|   |   |   └── SettingsPage.tsx
|   |   |
|   |   ├── hooks/            # Custom hooks
|   |   |   ├── useAuth.ts
|   |   |   ├── useWebSocket.ts
|   |   |   ├── useNotification.ts
|   |   |   └── useRealTimeData.ts
```

```
|   |   |
|   |   ├── context/          # React Context
|   |   |   ├── AuthContext.tsx
|   |   |   ├── ThemeContext.tsx
|   |   |   └── NotificationContext.tsx
|   |   |
|   |   ├── types/
|   |   |   └── index.d.ts
|   |   |
|   |   ├── utils/
|   |   |   ├── helpers.ts
|   |   |   └── constants.ts
|   |   |
|   |   ├── styles/
|   |   |   ├── globals.css
|   |   |   └── variables.css
|   |   |
|   |   ├── App.tsx
|   |   ├── main.tsx
|   |   └── vite-env.d.ts
|   |
|   ├── package.json
|   ├── vite.config.ts
|   └── tsconfig.json
|
├── mobile/                 # React Native + Expo
|   ├── src/
|   |   ├── screens/
|   |   |   ├── LoginScreen.tsx
|   |   |   ├── DashboardScreen.tsx
```

```
|   |   |   ├── AlertsScreen.tsx
|   |   |   └── MapScreen.tsx
|   |   |
|   |   ├── components/
|   |   ├── navigation/
|   |   ├── hooks/
|   |   └── api/
|   |
|   ├── app.json
|   ├── package.json
|   └── tsconfig.json
|
└── scripts/                # Utility scripts
    ├── generate-dataset.py     # Python script for data generation
    ├── train-model.py          # ML model training
    └── seed-database.ts        # Seed Supabase with test data
```

🔐 ENVIRONMENT VARIABLES

.env.example (Template)

bash

```
# ========================================
# SUPABASE CONFIGURATION
# ========================================
SUPABASE_URL=https://your-project.supabase.co
SUPABASE_ANON_KEY=your-anon-key-here
SUPABASE_SERVICE_KEY=your-service-role-key-here

# ========================================
# SERVER CONFIGURATION
# ========================================
NODE_ENV=development
```

```
PORT=5000
API_VERSION=v1
FRONTEND_URL=http://localhost:5173

# ==========================================
# JWT CONFIGURATION
# ==========================================
JWT_SECRET=your-super-secret-jwt-key-change-in-production
JWT_EXPIRES_IN=7d

# ==========================================
# EMAIL SERVICE (SMTP)
# ==========================================
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your-email@gmail.com
SMTP_PASSWORD=your-app-specific-password
SMTP_FROM=Kerala KSEBL Alerts <alerts@ksebl.gov.in>

# ==========================================
# SMS SERVICE (Twilio)
# ==========================================
TWILIO_ACCOUNT_SID=your-twilio-account-sid
TWILIO_AUTH_TOKEN=your-twilio-auth-token
TWILIO_PHONE_NUMBER=+1234567890

# ==========================================
# OPTIONAL: AI/ML SERVICES
# ==========================================
OPENAI_API_KEY=sk-your-openai-key
TENSORFLOW_MODEL_PATH=./backend/src/ml/model.json
```

```
# ==========================================
# LOGGING
# ==========================================
LOG_LEVEL=info
LOG_FILE=./logs/app.log

# ==========================================
# CORS
# ==========================================
ALLOWED_ORIGINS=http://localhost:5173,http://localhost:3000

# ==========================================
# RATE LIMITING
# ==========================================
RATE_LIMIT_WINDOW_MS=900000
RATE_LIMIT_MAX_REQUESTS=100
```

.env (Actual - NEVER COMMIT)

```
bash
# THIS FILE SHOULD BE IN .gitignore
# Copy from .env.example and fill with real values
```

📜 .cursorrules FILE

```
json
{
  "project": {
    "name": "Kerala LT Line Break Detection System",
    "description": "AI-powered real-time electrical line break detection for Kerala State Electricity Board",
    "techStack": {
```

```
    "backend": "Node.js with TypeScript",
    "frontend": "React.js with TypeScript + Vite",
    "mobile": "React Native with Expo",
    "database": "Supabase (PostgreSQL)",
    "deployment": "Vercel",
    "ml": "TensorFlow.js / Python (training)"
  }
},

"codeStandards": {
  "language": "TypeScript",
  "style": "Clean Code + SOLID Principles",
  "formatting": {
    "indentation": "2 spaces",
    "quotes": "single",
    "semicolons": true,
    "trailingComma": "es5",
    "printWidth": 100
  },
  "linting": "ESLint with TypeScript rules",
  "testing": "Jest + React Testing Library"
},

"architecture": {
  "pattern": "MVC + Service Layer",
  "principles": [
    "Separation of Concerns",
    "DRY (Don't Repeat Yourself)",
    "KISS (Keep It Simple, Stupid)",
    "Single Responsibility",
    "Dependency Injection"
```

```
  ],
  "folderStructure": "Feature-based + Layer-based hybrid"
},

"backend": {
  "framework": "Express.js",
  "apiStyle": "RESTful",
  "authentication": "JWT + Supabase Auth",
  "validation": "Zod or Joi",
  "errorHandling": "Centralized error middleware",
  "logging": "Winston or Pino",
  "testing": "Jest + Supertest"
},

"frontend": {
  "framework": "React 18+ with TypeScript",
  "buildTool": "Vite",
  "stateManagement": "React Context + Custom Hooks (avoid Redux for MVP)",
  "styling": "Tailwind CSS + CSS Modules",
  "routing": "React Router v6",
  "apiCalls": "Axios with interceptors",
  "charts": "Recharts or Chart.js",
  "maps": "Leaflet or Google Maps",
  "forms": "React Hook Form + Zod"
},

"database": {
  "orm": "Supabase Client (no ORM needed)",
  "migrations": "Supabase migrations",
  "seeding": "TypeScript seed scripts",
  "queryOptimization": "Use indexes, avoid N+1"
```

```
},

"security": {
  "secrets": "Environment variables only",
  "passwords": "bcrypt hashing (rounds: 12)",
  "cors": "Whitelist specific origins",
  "rateLimit": "Express rate limit middleware",
  "inputValidation": "All user inputs validated",
  "sqlInjection": "Parameterized queries only",
  "xss": "Sanitize all outputs"
},

"apiDesign": {
  "versioning": "/api/v1/...",
  "naming": "kebab-case for endpoints",
  "responseFormat": {
    "success": {
      "success": true,
      "data": {},
      "message": "string"
    },
    "error": {
      "success": false,
      "error": {
        "code": "ERROR_CODE",
        "message": "Human readable message"
      }
    }
  },
  "statusCodes": {
    "200": "OK",
```

```
    "201": "Created",
    "400": "Bad Request",
    "401": "Unauthorized",
    "403": "Forbidden",
    "404": "Not Found",
    "500": "Internal Server Error"
  },
  "pagination": {
    "page": "integer",
    "limit": "integer (max 100)",
    "total": "integer",
    "totalPages": "integer"
  }
},

"namingConventions": {
  "variables": "camelCase",
  "constants": "UPPER_SNAKE_CASE",
  "functions": "camelCase (verbs)",
  "classes": "PascalCase (nouns)",
  "interfaces": "PascalCase with 'I' prefix (optional)",
  "types": "PascalCase with 'T' prefix (optional)",
  "components": "PascalCase",
  "files": {
    "components": "PascalCase.tsx",
    "utilities": "camelCase.ts",
    "constants": "UPPER_SNAKE_CASE.ts",
    "types": "camelCase.types.ts"
  }
},
```

```
"comments": {
  "when": "Only when logic is complex or non-obvious",
  "style": "JSDoc for functions, inline for complex logic",
  "avoid": "Stating the obvious"
},

"gitWorkflow": {
  "branches": {
    "main": "Production-ready code",
    "develop": "Integration branch",
    "feature/*": "New features",
    "bugfix/*": "Bug fixes",
    "hotfix/*": "Critical production fixes"
  },
  "commits": {
    "format": "type(scope): message",
    "types": ["feat", "fix", "docs", "style", "refactor", "test", "chore"],
    "example": "feat(api): add line break detection endpoint"
  }
},

"deployment": {
  "platform": "Vercel",
  "environmentVariables": "Set in Vercel dashboard",
  "buildCommand": "npm run build",
  "outputDirectory": "dist or build",
  "autoDeployment": "Push to main branch"
},

"performance": {
  "backend": [
```

```
    "Use connection pooling",
    "Implement caching (Redis if needed)",
    "Optimize database queries",
    "Use compression middleware"
  ],
  "frontend": [
    "Code splitting",
    "Lazy loading components",
    "Memoization (React.memo, useMemo, useCallback)",
    "Optimize images (WebP, lazy loading)",
    "Minimize bundle size"
  ]
},

"errorHandling": {
  "backend": "Try-catch in async functions, centralized error middleware",
  "frontend": "Error boundaries for React components",
  "logging": "Log all errors with context"
},

"dataFlow": {
  "realTime": "WebSockets or Supabase Realtime subscriptions",
  "apiCalls": "Axios with error handling and retries",
  "stateUpdates": "Immutable patterns (spread operator, immer if needed)"
},

"mlIntegration": {
  "modelFormat": "TensorFlow.js SavedModel or ONNX",
  "inference": "Server-side (Node.js) or client-side (browser)",
  "preprocessing": "Feature extraction in backend service",
  "monitoring": "Track accuracy, latency, errors"
```

```
},

"documentation": {
  "api": "OpenAPI/Swagger specification",
  "code": "JSDoc for public functions",
  "readme": "Setup instructions, architecture overview",
  "deployment": "Step-by-step Vercel deployment guide"
},

"modularization": {
  "principles": [
    "Each module should have single responsibility",
    "Loose coupling between modules",
    "High cohesion within modules",
    "Use dependency injection",
    "Avoid circular dependencies"
  ]
},

"todoWorkflow": {
  "execution": "One task at a time, in order",
  "validation": "Test each task before moving to next",
  "documentation": "Update README after completing major features",
  "commit": "Commit after each completed task"
},

"cursor": {
  "projectLoading": "Load one project/feature at a time",
  "taskExecution": "Execute tasks sequentially from to-do list",
  "codeGeneration": "Follow all rules above",
  "refactoring": "Suggest improvements while maintaining functionality"
```

```
  },

  "bestPractices": [
    "Write self-documenting code",
    "Keep functions small (< 50 lines ideally)",
    "Use descriptive variable names",
    "Avoid magic numbers/strings (use constants)",
    "Handle edge cases",
    "Validate all inputs",
    "Never trust client data",
    "Use TypeScript strict mode",
    "Write tests for critical logic",
    "Keep dependencies minimal and updated",
    "Regular security audits (npm audit)",
    "Monitor application performance",
    "Use environment variables for configuration",
    "Never commit secrets to Git"
  ]
}
```

🔌 REST API ENDPOINTS

Authentication

```
typescript
POST   /api/v1/auth/register
POST   /api/v1/auth/login
POST   /api/v1/auth/logout
POST   /api/v1/auth/refresh-token
GET    /api/v1/auth/me
PUT    /api/v1/auth/change-password
```

Substations

typescript

```
GET    /api/v1/substations            // List all substations
GET    /api/v1/substations/:id        // Get single substation
POST   /api/v1/substations            // Create substation (admin only)
PUT    /api/v1/substations/:id        // Update substation (admin only)
DELETE /api/v1/substations/:id        // Delete substation (admin only)
GET    /api/v1/substations/:id/feeders  // Get all feeders in substation
```

Feeders

typescript

```
GET    /api/v1/feeders                // List all feeders
GET    /api/v1/feeders/:id            // Get single feeder
POST   /api/v1/feeders                // Create feeder (admin only)
PUT    /api/v1/feeders/:id            // Update feeder (admin only)
DELETE /api/v1/feeders/:id            // Delete feeder (admin only)
GET    /api/v1/feeders/:id/events     // Get events for feeder
GET    /api/v1/feeders/:id/health     // Get real-time health status
```

Line Break Events

typescript

```
GET    /api/v1/events                 // List all events (paginated)
GET    /api/v1/events/:id             // Get single event
POST   /api/v1/events                 // Create event (system generated)
PUT    /api/v1/events/:id             // Update event (status, notes)
DELETE /api/v1/events/:id             // Delete event (admin only)
GET    /api/v1/events/recent          // Get recent events (last 24h)
GET    /api/v1/events/active          // Get active (unresolved) events
POST   /api/v1/events/:id/acknowledge   // Acknowledge event
POST   /api/v1/events/:id/assign      // Assign to field crew
POST   /api/v1/events/:id/resolve     // Mark as resolved
```

Waveform Data

typescript

```
GET    /api/v1/waveforms              // List waveforms (paginated)
GET    /api/v1/waveforms/:id          // Get single waveform
POST   /api/v1/waveforms              // Upload waveform data
POST   /api/v1/waveforms/analyze      // Analyze waveform (ML inference)
GET    /api/v1/waveforms/:id/features // Get extracted features
```

Dashboard & Analytics

typescript

```
GET    /api/v1/dashboard/summary      // Dashboard summary stats
GET    /api/v1/dashboard/map-data     // Data for map visualization
GET    /api/v1/dashboard/recent-events // Recent events for timeline
GET    /api/v1/analytics/trends       // Historical trends
GET    /api/v1/analytics/performance  // Model performance metrics
GET    /api/v1/analytics/reports      // Generate reports
```

Alerts

typescript

```
GET    /api/v1/alerts                 // List all alerts
GET    /api/v1/alerts/:id             // Get single alert
POST   /api/v1/alerts                 // Send manual alert
GET    /api/v1/alerts/user/:userId    // Get alerts for user
PUT    /api/v1/alerts/:id/status      // Update alert status
```

Users

typescript

```
GET    /api/v1/users                  // List all users (admin only)
GET    /api/v1/users/:id              // Get single user
POST   /api/v1/users                  // Create user (admin only)
PUT    /api/v1/users/:id              // Update user
DELETE /api/v1/users/:id              // Delete user (admin only)
```

PUT /api/v1/users/:id/activate // Activate/deactivate user

GET /api/v1/users/field-crew // Get all field crew members

ML Model Management

typescript

POST /api/v1/ml/predict // Real-time prediction

POST /api/v1/ml/batch-predict // Batch prediction

GET /api/v1/ml/models // List available models

POST /api/v1/ml/models/upload // Upload new model (admin)

GET /api/v1/ml/metrics // Get model performance metrics

POST /api/v1/ml/retrain // Trigger model retraining

System

typescript

GET /api/v1/health // Health check

GET /api/v1/logs // System logs (admin only)

GET /api/v1/stats // System statistics

API REQUEST/RESPONSE EXAMPLES

Example 1: User Login

Request:

typescript

POST /api/v1/auth/login

Content-Type: application/json

{

"email": "operator@ksebl.gov.in",

"password": "SecurePass123!"

}

Response:

typescript

200 OK

Content-Type: application/json


```
{
  "success": true,
  "data": {
    "user": {
      "id": "550e8400-e29b-41d4-a716-446655440000",
      "email": "operator@ksebl.gov.in",
      "full_name": "John Doe",
      "role": "operator",
      "phone": "+919876543210"
    },
    "tokens": {
      "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
      "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
      "expiresIn": 604800
    }
  },
  "message": "Login successful"
}
```


Example 2: Detect Line Break

Request:

typescript

POST /api/v1/waveforms/analyze

Content-Type: application/json

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...


```
{
  "feeder_id": "660e8400-e29b-41d4-a716-446655440001",
  "timestamp": "2025-10-03T14:30:45Z",
  "waveform_data": {
    "current_r": [45.2, 45.3, 45.1, ..., 18.5, 18.2],
    "current_y": [44.8, 45.0, 44.9, ..., 18.1, 17.9],
    "current_b": [45.5, 45.6, 45.4, ..., 18.8, 18.6],
    "voltage_r": [230.5, 230.6, ..., 195.2, 195.0],
    "voltage_y": [229.8, 230.0, ..., 194.8, 194.5],
    "voltage_b": [230.2, 230.3, ..., 195.5, 195.3]
  },
  "sampling_rate": 10000,
  "duration_seconds": 4
}
```


Response:

typescript

200 OK

Content-Type: application/json


```
{
  "success": true,
  "data": {
    "prediction": {
      "fault_detected": true,
      "fault_type": "LINE_BREAK",
      "confidence": 0.9687,
      "estimated_location_km": 2.45,
```

```
    "severity": "high",
    "detection_time_ms": 167
   },
   "event": {
    "id": "770e8400-e29b-41d4-a716-446655440002",
    "feeder_id": "660e8400-e29b-41d4-a716-446655440001",
    "detected_at": "2025-10-03T14:30:45.167Z",
    "status": "detected",
    "breaker_tripped": true,
    "trip_time_ms": 245
   },
   "alerts_sent": [
    {
     "type": "sms",
     "recipient": "+919876543210",
     "status": "sent"
    },
    {
     "type": "email",
     "recipient": "fieldcrew@ksebl.gov.in",
     "status": "sent"
    }
   ]
  },
  "message": "Line break detected and breaker tripped successfully"
}
```


Example 3: Get Dashboard Summary

Request:

typescript

GET /api/v1/dashboard/summary

Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...

Response:

typescript

200 OK

Content-Type: application/json


```
{
  "success": true,
  "data": {
    "overview": {
      "total_substations": 45,
      "active_feeders": 342,
      "total_events_today": 8,
      "active_events": 2,
      "resolved_events": 6,
      "model_accuracy": 0.9687
    },
    "recent_events": [
      {
        "id": "770e8400-e29b-41d4-a716-446655440002",
        "feeder_name": "Trivandrum North-F12",
        "detected_at": "2025-10-03T14:30:45Z",
        "status": "crew_dispatched",
        "severity": "high",
        "location_km": 2.45
      },
      {
        "id": "770e8400-e29b-41d4-a716-446655440003",
```

```
      "feeder_name": "Kochi East-F07",
      "detected_at": "2025-10-03T13:15:20Z",
      "status": "resolved",
      "severity": "medium",
      "location_km": 3.82
    }
  ],
  "alerts": {
    "critical": 2,
    "high": 5,
    "medium": 12,
    "low": 8
  },
  "performance": {
    "avg_detection_time_ms": 185,
    "avg_response_time_minutes": 12.5,
    "false_positive_rate": 0.018
  }
 },
 "message": "Dashboard data retrieved successfully"
}
```


🛠️ BACKEND IMPLEMENTATION (TypeScript)

1. Supabase Configuration

typescript

```
// backend/src/config/supabase.ts

import { createClient } from '@supabase/supabase-js';
import { config } from './env';

if (!config.supabase.url || !config.supabase.serviceKey) {
```

```
  throw new Error('Missing Supabase configuration');
}

export const supabase = createClient(
  config.supabase.url,
  config.supabase.serviceKey,
  {
    auth: {
      autoRefreshToken: true,
      persistSession: true,
    },
  }
);

export const supabaseAdmin = createClient(
  config.supabase.url,
  config.supabase.serviceKey
);
```

2. Environment Configuration

typescript

```
// backend/src/config/env.ts

import dotenv from 'dotenv';
import { z } from 'zod';

dotenv.config();

const envSchema = z.object({
  NODE_ENV: z.enum(['development', 'production', 'test']).default('development'),
  PORT: z.string().transform(Number).default('5000'),
```

```
  // Supabase
  SUPABASE_URL: z.string().url(),
  SUPABASE_ANON_KEY: z.string(),
  SUPABASE_SERVICE_KEY: z.string(),

  // JWT
  JWT_SECRET: z.string().min(32),
  JWT_EXPIRES_IN: z.string().default('7d'),

  // Email
  SMTP_HOST: z.string(),
  SMTP_PORT: z.string().transform(Number),
  SMTP_USER: z.string().email(),
  SMTP_PASSWORD: z.string(),
  SMTP_FROM: z.string(),

  // SMS
  TWILIO_ACCOUNT_SID: z.string().optional(),
  TWILIO_AUTH_TOKEN: z.string().optional(),
  TWILIO_PHONE_NUMBER: z.string().optional(),

  // Other
  FRONTEND_URL: z.string().url(),
  LOG_LEVEL: z.enum(['error', 'warn', 'info', 'debug']).default('info'),
});

const parseEnv = () => {
  try {
    return envSchema.parse(process.env);
  } catch (error) {
```

```
    console.error('❌ Invalid environment variables:', error);
    process.exit(1);
  }
};

const env = parseEnv();

export const config = {
  env: env.NODE_ENV,
  port: env.PORT,
  supabase: {
    url: env.SUPABASE_URL,
    anonKey: env.SUPABASE_ANON_KEY,
    serviceKey: env.SUPABASE_SERVICE_KEY,
  },
  jwt: {
    secret: env.JWT_SECRET,
    expiresIn: env.JWT_EXPIRES_IN,
  },
  smtp: {
    host: env.SMTP_HOST,
    port: env.SMTP_PORT,
    user: env.SMTP_USER,
    password: env.SMTP_PASSWORD,
    from: env.SMTP_FROM,
  },
  twilio: {
    accountSid: env.TWILIO_ACCOUNT_SID,
    authToken: env.TWILIO_AUTH_TOKEN,
    phoneNumber: env.TWILIO_PHONE_NUMBER,
  },
```

```
  frontend: {
    url: env.FRONTEND_URL,
  },
  logging: {
    level: env.LOG_LEVEL,
  },
};
```

3. Auth Middleware

typescript

```
// backend/src/middleware/auth.middleware.ts

import { Request, Response, NextFunction } from 'express';
import jwt from 'jsonwebtoken';
import { config } from '../config/env';
import { supabase } from '../config/supabase';
import { ApiError } from '../utils/ApiError';

export interface AuthRequest extends Request {
  user?: {
    id: string;
    email: string;
    role: string;
  };
}

export const authenticate = async (
  req: AuthRequest,
  res: Response,
  next: NextFunction
) => {
```

```
try {
  const authHeader = req.headers.authorization;

  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    throw new ApiError(401, 'No token provided');
  }

  const token = authHeader.split(' ')[1];

  // Verify JWT
  const decoded = jwt.verify(token, config.jwt.secret) as {
    userId: string;
    email: string;
    role: string;
  };

  // Verify user still exists and is active
  const { data: user, error } = await supabase
    .from('users')
    .select('id, email, role, is_active')
    .eq('id', decoded.userId)
    .single();

  if (error || !user || !user.is_active) {
    throw new ApiError(401, 'Invalid or expired token');
  }

  req.user = {
    id: user.id,
    email: user.email,
    role: user.role,
```

```
    };

    next();
  } catch (error) {
    if (error instanceof jwt.JsonWebTokenError) {
      next(new ApiError(401, 'Invalid token'));
    } else {
      next(error);
    }
  }
};

export const authorize = (...roles: string[]) => {
  return (req: AuthRequest, res: Response, next: NextFunction) => {
    if (!req.user) {
      return next(new ApiError(401, 'Not authenticated'));
    }

    if (!roles.includes(req.user.role)) {
      return next(new ApiError(403, 'Insufficient permissions'));
    }

    next();
  };
};
```

4. Event Controller

typescript

```
// backend/src/controllers/eventController.ts

import { Response, NextFunction } from 'express';
```

```
import { AuthRequest } from '../middleware/auth.middleware';
import { supabase } from '../config/supabase';
import { ApiError } from '../utils/ApiError';
import { ApiResponse } from '../utils/ApiResponse';
import { alertService } from '../services/alertService';
import { z } from 'zod';

const createEventSchema = z.object({
  feeder_id: z.string().uuid(),
  confidence_score: z.number().min(0).max(1),
  estimated_location_km: z.number().positive(),
  fault_type: z.string().default('LINE_BREAK'),
  severity: z.enum(['low', 'medium', 'high', 'critical']),
});

export class EventController {
  async getAllEvents(req: AuthRequest, res: Response, next: NextFunction) {
    try {
      const page = parseInt(req.query.page as string) || 1;
      const limit = Math.min(parseInt(req.query.limit as string) || 20, 100);
      const offset = (page - 1) * limit;
      const status = req.query.status as string;

      let query = supabase
        .from('line_break_events')
        .select(`
          *,
          feeders(name, code, substations(name))
        `, { count: 'exact' })
        .order('detected_at', { ascending: false })
        .range(offset, offset + limit - 1);
```

```
    if (status) {
      query = query.eq('status', status);
    }

    const { data, error, count } = await query;

    if (error) throw error;

    return res.json(
      new ApiResponse(true, {
        events: data,
        pagination: {
          page,
          limit,
          total: count || 0,
          totalPages: Math.ceil((count || 0) / limit),
        },
      })
    );
  } catch (error) {
    next(error);
  }
}

async getEventById(req: AuthRequest, res: Response, next: NextFunction) {
  try {
    const { id } = req.params;

    const { data, error } = await supabase
      .from('line_break_events')
```

```
      .select(`
        *,
        feeders(
          *,
          substations(*)
        ),
        users(full_name, email, phone)
      `)
      .eq('id', id)
      .single();

    if (error || !data) {
      throw new ApiError(404, 'Event not found');
    }

    return res.json(new ApiResponse(true, { event: data }));
  } catch (error) {
    next(error);
  }
}

async createEvent(req: AuthRequest, res: Response, next: NextFunction) {
  try {
    const validatedData = createEventSchema.parse(req.body);

    const eventData = {
      ...validatedData,
      detected_at: new Date().toISOString(),
      detection_method: 'ai_model',
      status: 'detected',
      breaker_tripped: true,
```

```
      trip_time_ms: Math.floor(Math.random() * 100) + 150, // Simulate trip time
    };

    const { data: event, error } = await supabase
      .from('line_break_events')
      .insert(eventData)
      .select()
      .single();

    if (error) throw error;

    // Send alerts asynchronously
    alertService.sendEventAlerts(event).catch(console.error);

    return res.status(201).json(
      new ApiResponse(
        true,
        { event },
        'Line break event created and alerts sent'
      )
    );
  } catch (error) {
    next(error);
  }
}

async acknowledgeEvent(req: AuthRequest, res: Response, next: NextFunction) {
  try {
    const { id } = req.params;
    const userId = req.user!.id;
```

```
    const { data, error } = await supabase
      .from('line_break_events')
      .update({
        status: 'acknowledged',
        assigned_to: userId,
        updated_at: new Date().toISOString(),
      })
      .eq('id', id)
      .select()
      .single();

    if (error || !data) {
      throw new ApiError(404, 'Event not found');
    }

    return res.json(
      new ApiResponse(true, { event: data }, 'Event acknowledged')
    );
  } catch (error) {
    next(error);
  }
}

async resolveEvent(req: AuthRequest, res: Response, next: NextFunction) {
  try {
    const { id } = req.params;
    const { resolution_notes } = req.body;

    const { data, error } = await supabase
      .from('line_break_events')
      .update({
```

```
        status: 'resolved',
        resolved_at: new Date().toISOString(),
        resolution_notes,
      })
      .eq('id', id)
      .select()
      .single();

    if (error || !data) {
      throw new ApiError(404, 'Event not found');
    }

    return res.json(
      new ApiResponse(true, { event: data }, 'Event resolved successfully')
    );
  } catch (error) {
    next(error);
  }
}

async getActiveEvents(req: AuthRequest, res: Response, next: NextFunction) {
  try {
    const { data, error } = await supabase
      .from('line_break_events')
      .select(`
        *,
        feeders(name, code, substations(name))
      `)
      .in('status', ['detected', 'acknowledged', 'crew_dispatched'])
      .order('detected_at', { ascending: false });
```

```
    if (error) throw error;

    return res.json(new ApiResponse(true, { events: data }));
   } catch (error) {
    next(error);
   }
  }
}

export const eventController = new EventController();
```

5. ML Service

```
typescript
// backend/src/services/mlService.ts

import * as tf from '@tensorflow/tfjs-node';
import { FeatureExtractor } from './featureExtractor';
import path from 'path';

interface WaveformData {
  current_r: number[];
  current_y: number[];
  current_b: number[];
  voltage_r: number[];
  voltage_y: number[];
  voltage_b: number[];
}

interface PredictionResult {
  fault_detected: boolean;
  fault_type: string;
```

```
  confidence: number;
  estimated_location_km: number;
  severity: 'low' | 'medium' | 'high' | 'critical';
  detection_time_ms: number;
}

class MLService {
  private model: tf.LayersModel | null = null;
  private scaler: any = null;
  private featureExtractor: FeatureExtractor;
  private readonly MODEL_PATH = path.join(__dirname, '../ml/model.json');
  private readonly SCALER_PATH = path.join(__dirname, '../ml/scaler.json');

  constructor() {
    this.featureExtractor = new FeatureExtractor();
    this.loadModel();
  }

  private async loadModel() {
    try {
      console.log('Loading ML model...');
      this.model = await tf.loadLayersModel(`file://${this.MODEL_PATH}`);

      // Load scaler parameters
      const scalerData = require(this.SCALER_PATH);
      this.scaler = scalerData;

      console.log('✅ ML model loaded successfully');
    } catch (error) {
      console.error('❌ Failed to load ML model:', error);
      console.log('⚠️  Running in simulation mode');
```

```
  }
}

async predict(waveformData: WaveformData): Promise<PredictionResult> {
  const startTime = Date.now();

  try {
    // Extract features
    const features = this.featureExtractor.extractFeatures(waveformData);

    let prediction;

    if (this.model && this.scaler) {
      // Real ML prediction
      const scaledFeatures = this.scaleFeatures(features);
      const tensor = tf.tensor2d([scaledFeatures]);
      const output = this.model.predict(tensor) as tf.Tensor;
      const probabilities = await output.data();

      prediction = {
        class: this.getClassFromProbabilities(Array.from(probabilities)),
        confidence: Math.max(...Array.from(probabilities)),
      };

      tf.dispose([tensor, output]);
    } else {
      // Simulation mode - rule-based detection
      prediction = this.simulateDetection(features);
    }

    const detectionTime = Date.now() - startTime;
```

```
    return {
      fault_detected: prediction.class === 'LINE_BREAK',
      fault_type: prediction.class,
      confidence: prediction.confidence,
      estimated_location_km: this.estimateLocation(waveformData),
      severity: this.determineSeverity(prediction.confidence),
      detection_time_ms: detectionTime,
    };
  } catch (error) {
    console.error('Prediction error:', error);
    throw error;
  }
}

private scaleFeatures(features: number[]): number[] {
  if (!this.scaler) return features;

  return features.map((value, index) => {
    const mean = this.scaler.mean[index] || 0;
    const std = this.scaler.std[index] || 1;
    return (value - mean) / std;
  });
}

private getClassFromProbabilities(probabilities: number[]): string {
  const classes = ['NORMAL', 'LINE_BREAK', 'SHORT_CIRCUIT', 'OVERLOAD'];
  const maxIndex = probabilities.indexOf(Math.max(...probabilities));
  return classes[maxIndex];
}
```

```
  private simulateDetection(features: number[]): { class: string; confidence: number } {
    // Simple rule-based detection for demo
    const currentDropR = features[0]; // Assuming first feature is current RMS
    const negativeSequence = features[features.length - 2];

    if (currentDropR < 30 && negativeSequence > 5) {
      return { class: 'LINE_BREAK', confidence: 0.85 + Math.random() * 0.15 };
    } else if (currentDropR > 100) {
      return { class: 'SHORT_CIRCUIT', confidence: 0.90 + Math.random() * 0.10 };
    } else if (currentDropR > 70) {
      return { class: 'OVERLOAD', confidence: 0.75 + Math.random() * 0.15 };
    } else {
      return { class: 'NORMAL', confidence: 0.95 + Math.random() * 0.05 };
    }
  }

  private estimateLocation(waveformData: WaveformData): number {
    // Simplified impedance-based method
    const avgCurrentDrop = (
      Math.abs(waveformData.current_r[0] - waveformData.current_r[waveformData.current_r.length -
1]) +
      Math.abs(waveformData.current_y[0] - waveformData.current_y[waveformData.current_y.length
- 1]) +
      Math.abs(waveformData.current_b[0] -
waveformData.current_b[waveformData.current_b.length - 1])
    ) / 3;

    const avgVoltageDrop = (
      Math.abs(waveformData.voltage_r[0] - waveformData.voltage_r[waveformData.voltage_r.length -
1]) +
      Math.abs(waveformData.voltage_y[0] - waveformData.voltage_y[waveformData.voltage_y.length
- 1]) +
```

```
    Math.abs(waveformData.voltage_b[0] - waveformData.voltage_b[waveformData.voltage_b.length
- 1])
   ) / 3;

   // Simplified calculation (in reality, use line impedance data)
   const estimatedDistance = (avgVoltageDrop / avgCurrentDrop) * 2.5;

   return Math.max(0.5, Math.min(estimatedDistance, 5.0)); // Clamp between 0.5-5 km
  }

  private determineSeverity(confidence: number): 'low' | 'medium' | 'high' | 'critical' {
   if (confidence >= 0.95) return 'critical';
   if (confidence >= 0.85) return 'high';
   if (confidence >= 0.70) return 'medium';
   return 'low';
  }
}

export const mlService = new MLService();
```

6. Alert Service

```
typescript
// backend/src/services/alertService.ts

import nodemailer from 'nodemailer';
import twilio from 'twilio';
import { config } from '../config/env';
import { supabase } from '../config/supabase';

interface LineBreakEvent {
  id: string;
```

```
  feeder_id: string;
  detected_at: string;
  estimated_location_km: number;
  severity: string;
  confidence_score: number;
}

class AlertService {
  private emailTransporter: nodemailer.Transporter;
  private twilioClient: any;

  constructor() {
    // Initialize email transporter
    this.emailTransporter = nodemailer.createTransporter({
      host: config.smtp.host,
      port: config.smtp.port,
      secure: config.smtp.port === 465,
      auth: {
        user: config.smtp.user,
        pass: config.smtp.password,
      },
    });

    // Initialize Twilio (if configured)
    if (config.twilio.accountSid && config.twilio.authToken) {
      this.twilioClient = twilio(
        config.twilio.accountSid,
        config.twilio.authToken
      );
    }
  }
```

```
async sendEventAlerts(event: LineBreakEvent) {
  try {
    // Get feeder and substation details
    const { data: feederData } = await supabase
      .from('feeders')
      .select('name, code, substations(name)')
      .eq('id', event.feeder_id)
      .single();

    if (!feederData) {
      console.error('Feeder not found for alert');
      return;
    }

    // Get users to notify (operators and field crew)
    const { data: users } = await supabase
      .from('users')
      .select('id, full_name, email, phone, role')
      .eq('is_active', true)
      .in('role', ['operator', 'field_crew']);

    if (!users || users.length === 0) {
      console.log('No users to notify');
      return;
    }

    const alertPromises = users.map(async (user) => {
      const alerts = [];

      // Send email
```

```
      if (user.email) {
        alerts.push(
          this.sendEmailAlert(user, event, feederData)
        );
      }

      // Send SMS to field crew
      if (user.role === 'field_crew' && user.phone) {
        alerts.push(
          this.sendSMSAlert(user, event, feederData)
        );
      }

      return Promise.allSettled(alerts);
    });

    await Promise.all(alertPromises);
    console.log(`✅ Alerts sent for event ${event.id}`);
  } catch (error) {
    console.error('Error sending alerts:', error);
  }
}

private async sendEmailAlert(user: any, event: LineBreakEvent, feederData: any) {
  try {
    const subject = `🚨 LINE BREAK DETECTED - ${feederData.substations.name}`;

    const html = `
      <div style="font-family: Arial, sans-serif; max-width: 600px; margin: 0 auto;">
        <h2 style="color: #dc2626;">⚠️ Line Break Alert</h2>
```

```
<div style="background: #fee2e2; padding: 15px; border-radius: 8px; margin: 20px 0;">
  <p style="margin: 0; font-weight: bold;">URGENT: Line break detected</p>
</div>

<table style="width: 100%; border-collapse: collapse;">
  <tr>
    <td style="padding: 10px; border-bottom: 1px solid #ddd;"><strong>Substation:</strong></td>
    <td style="padding: 10px; border-bottom: 1px solid #ddd;">${feederData.substations.name}</td>
  </tr>
  <tr>
    <td style="padding: 10px; border-bottom: 1px solid #ddd;"><strong>Feeder:</strong></td>
    <td style="padding: 10px; border-bottom: 1px solid #ddd;">${feederData.name} (${feederData.code})</td>
  </tr>
  <tr>
    <td style="padding: 10px; border-bottom: 1px solid #ddd;"><strong>Location:</strong></td>
    <td style="padding: 10px; border-bottom: 1px solid #ddd;">${event.estimated_location_km.toFixed(2)} km from substation</td>
  </tr>
  <tr>
    <td style="padding: 10px; border-bottom: 1px solid #ddd;"><strong>Severity:</strong></td>
    <td style="padding: 10px; border-bottom: 1px solid #ddd; text-transform: uppercase;">${event.severity}</td>
  </tr>
  <tr>
    <td style="padding: 10px; border-bottom: 1px solid #ddd;"><strong>Confidence:</strong></td>
    <td style="padding: 10px; border-bottom: 1px solid #ddd;">${(event.confidence_score * 100).toFixed(1)}%</td>
  </tr>
  <tr>
```

```
          <td style="padding: 10px; border-bottom: 1px solid #ddd;"><strong>Detected At:</strong></td>
          <td style="padding: 10px; border-bottom: 1px solid #ddd;">${new Date(event.detected_at).toLocaleString('en-IN')}</td>
        </tr>
      </table>

      <div style="margin-top: 20px; padding: 15px; background: #fef3c7; border-radius: 8px;">
        <p style="margin: 0;"><strong>Action Required:</strong></p>
<p style="margin: 5px 0 0 0;">Please acknowledge this alert and dispatch field crew immediately.</p> </div>

    <div style="margin-top: 20px; text-align: center;">
      <a href="${config.frontend.url}/events/${event.id}"
        style="background: #dc2626; color: white; padding: 12px 24px; text-decoration: none; border-radius: 6px; display: inline-block;">
        View Event Details
      </a>
    </div>

    <div style="margin-top: 30px; padding-top: 20px; border-top: 1px solid #ddd; color: #666; font-size: 12px;">
      <p>This is an automated alert from Kerala LT Line Break Detection System.</p>
      <p>Kerala State Electricity Board Limited (KSEBL)</p>
    </div>
  </div>
`;

 const info = await this.emailTransporter.sendMail({
  from: config.smtp.from,
  to: user.email,
  subject,
  html,
```

```
  });

  // Log alert to database
  await supabase.from('alerts').insert({
    event_id: event.id,
    alert_type: 'email',
    recipient_id: user.id,
    recipient_contact: user.email,
    message: subject,
    sent_at: new Date().toISOString(),
    status: 'sent',
  });

  return info;
} catch (error) {
  console.error('Email send error:', error);

  // Log failed alert
  await supabase.from('alerts').insert({
    event_id: event.id,
    alert_type: 'email',
    recipient_id: user.id,
    recipient_contact: user.email,
    status: 'failed',
    error_message: error instanceof Error ? error.message : 'Unknown error',
  });

  throw error;
}
}
```

```
private async sendSMSAlert(user: any, event: LineBreakEvent, feederData: any) {
if (!this.twilioClient) {
console.log('SMS service not configured');
return;
}


try {
  const message = `
🚨 KSEBL LINE BREAK ALERT

Substation: ${feederData.substations.name}
Feeder: ${feederData.code}
Location: ${event.estimated_location_km.toFixed(2)} km
Severity: ${event.severity.toUpperCase()}
Time: ${new Date(event.detected_at).toLocaleTimeString('en-IN')}

Immediate action required! View:
c
o
n
f
i
g
.
f
r
o
n
t
e
```

n

d

.

u

r

l

/

e

v

e

n

t

s

/

config.frontend.url/events/{event.id} `.trim();

```
  const sms = await this.twilioClient.messages.create({
    body: message,
    from: config.twilio.phoneNumber,
    to: user.phone,
  });

  // Log alert to database
  await supabase.from('alerts').insert({
    event_id: event.id,
    alert_type: 'sms',
    recipient_id: user.id,
    recipient_contact: user.phone,
    message,
    sent_at: new Date().toISOString(),
```

```
    status: 'sent',
  });

  return sms;
 } catch (error) {
  console.error('SMS send error:', error);

  // Log failed alert
  await supabase.from('alerts').insert({
    event_id: event.id,
    alert_type: 'sms',
    recipient_id: user.id,
    recipient_contact: user.phone,
    status: 'failed',
    error_message: error instanceof Error ? error.message : 'Unknown error',
  });

  throw error;
 }
 }
 }

export const alertService = new AlertService();
```

7. Express Server

```typescript
// backend/src/server.ts

import express, { Application } from 'express';
```

```
import cors from 'cors';
import helmet from 'helmet';
import compression from 'compression';
import morgan from 'morgan';
import rateLimit from 'express-rate-limit';
import { config } from './config/env';
import { errorHandler } from './middleware/error.middleware';

// Routes
import authRoutes from './routes/auth.routes';
import substationRoutes from './routes/substation.routes';
import feederRoutes from './routes/feeder.routes';
import eventRoutes from './routes/event.routes';
import waveformRoutes from './routes/waveform.routes';
import dashboardRoutes from './routes/dashboard.routes';
import userRoutes from './routes/user.routes';

const app: Application = express();

// Security middleware
app.use(helmet());
app.use(
  cors({
    origin: config.frontend.url,
    credentials: true,
  })
);

// Rate limiting
const limiter = rateLimit({
  windowMs: 15 * 60 * 1000, // 15 minutes
```

```
  max: 100, // limit each IP to 100 requests per windowMs
  message: 'Too many requests from this IP, please try again later.',
});
app.use('/api/', limiter);

// Body parsing
app.use(express.json({ limit: '10mb' }));
app.use(express.urlencoded({ extended: true, limit: '10mb' }));

// Compression
app.use(compression());

// Logging
if (config.env === 'development') {
  app.use(morgan('dev'));
} else {
  app.use(morgan('combined'));
}

// Health check
app.get('/health', (req, res) => {
  res.json({
    status: 'ok',
    timestamp: new Date().toISOString(),
    uptime: process.uptime(),
    environment: config.env,
  });
});

// API Routes
const API_PREFIX = `/api/${config.apiVersion || 'v1'}`;
```

```
app.use(`${API_PREFIX}/auth`, authRoutes);
app.use(`${API_PREFIX}/substations`, substationRoutes);
app.use(`${API_PREFIX}/feeders`, feederRoutes);
app.use(`${API_PREFIX}/events`, eventRoutes);
app.use(`${API_PREFIX}/waveforms`, waveformRoutes);
app.use(`${API_PREFIX}/dashboard`, dashboardRoutes);
app.use(`${API_PREFIX}/users`, userRoutes);

// 404 handler
app.use('*', (req, res) => {
  res.status(404).json({
    success: false,
    error: {
      code: 'NOT_FOUND',
      message: `Route ${req.originalUrl} not found`,
    },
  });
});

// Error handler (must be last)
app.use(errorHandler);

// Start server
const PORT = config.port || 5000;

app.listen(PORT, () => {
  console.log('');
  console.log('🚀 ========================================');
  console.log(`   Kerala Line Break Detection System`);
  console.log('   ========================================');
```

```
  console.log(`   Environment: ${config.env}`);
  console.log(`   Server: http://localhost:${PORT}`);
  console.log(`   API Base: ${API_PREFIX}`);
  console.log(`   Health: http://localhost:${PORT}/health`);
  console.log('   ========================================');
  console.log('');
});

export default app;
```

💻 FRONTEND IMPLEMENTATION (React + TypeScript)

1. API Client Setup

typescript

```
// frontend/src/api/client.ts

import axios, { AxiosError, AxiosRequestConfig, AxiosResponse } from 'axios';

const API_BASE_URL = import.meta.env.VITE_API_URL || 'http://localhost:5000/api/v1';

export const apiClient = axios.create({
  baseURL: API_BASE_URL,
  timeout: 30000,
  headers: {
    'Content-Type': 'application/json',
  },
});

// Request interceptor - Add auth token
apiClient.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('accessToken');
```

```
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  },
  (error) => Promise.reject(error)
);

// Response interceptor - Handle errors globally
apiClient.interceptors.response.use(
  (response: AxiosResponse) => response.data,
  async (error: AxiosError) => {
    const originalRequest = error.config as AxiosRequestConfig & { _retry?: boolean };

    // Handle 401 - Token expired
    if (error.response?.status === 401 && !originalRequest._retry) {
      originalRequest._retry = true;

      try {
        const refreshToken = localStorage.getItem('refreshToken');
        const { data } = await axios.post(`${API_BASE_URL}/auth/refresh-token`, {
          refreshToken,
        });

        localStorage.setItem('accessToken', data.accessToken);

        if (originalRequest.headers) {
          originalRequest.headers.Authorization = `Bearer ${data.accessToken}`;
        }

        return apiClient(originalRequest);
```

```
      } catch (refreshError) {
        // Refresh failed - logout user
        localStorage.removeItem('accessToken');
        localStorage.removeItem('refreshToken');
        window.location.href = '/login';
        return Promise.reject(refreshError);
      }
    }

    return Promise.reject(error);
  }
);

export interface ApiResponse<T = any> {
  success: boolean;
  data?: T;
  error?: {
    code: string;
    message: string;
  };
  message?: string;
}
```

2. Event API

typescript

```
// frontend/src/api/event.api.ts

import { apiClient, ApiResponse } from './client';

export interface LineBreakEvent {
  id: string;
```

```
  feeder_id: string;
  detected_at: string;
  confidence_score: number;
  estimated_location_km: number;
  fault_type: string;
  severity: 'low' | 'medium' | 'high' | 'critical';
  status: 'detected' | 'acknowledged' | 'crew_dispatched' | 'resolved';
  breaker_tripped: boolean;
  trip_time_ms: number;
  resolved_at?: string;
  resolution_notes?: string;
  assigned_to?: string;
  feeder?: {
    name: string;
    code: string;
    substation?: {
      name: string;
    };
  };
}

export interface EventsListResponse {
  events: LineBreakEvent[];
  pagination: {
    page: number;
    limit: number;
    total: number;
    totalPages: number;
  };
}
```

```
class EventAPI {
  async getEvents(params?: {
    page?: number;
    limit?: number;
    status?: string;
  }): Promise<EventsListResponse> {
    const response = await apiClient.get<ApiResponse<EventsListResponse>>('/events', {
      params,
    });
    return response.data!;
  }

  async getEventById(id: string): Promise<LineBreakEvent> {
    const response = await apiClient.get<ApiResponse<{ event: LineBreakEvent }>>(
      `/events/${id}`
    );
    return response.data!.event;
  }

  async getActiveEvents(): Promise<LineBreakEvent[]> {
    const response = await apiClient.get<ApiResponse<{ events: LineBreakEvent[] }>>(
      '/events/active'
    );
    return response.data!.events;
  }

  async acknowledgeEvent(id: string): Promise<LineBreakEvent> {
    const response = await apiClient.post<ApiResponse<{ event: LineBreakEvent }>>(
      `/events/${id}/acknowledge`
    );
    return response.data!.event;
```

```
  }

  async assignEvent(id: string, userId: string): Promise<LineBreakEvent> {
    const response = await apiClient.post<ApiResponse<{ event: LineBreakEvent }>>(
      `/events/${id}/assign`,
      { userId }
    );
    return response.data!.event;
  }

  async resolveEvent(id: string, notes: string): Promise<LineBreakEvent> {
    const response = await apiClient.post<ApiResponse<{ event: LineBreakEvent }>>(
      `/events/${id}/resolve`,
      { resolution_notes: notes }
    );
    return response.data!.event;
  }
}

export const eventAPI = new EventAPI();
```

3. Dashboard Page

```
typescript
// frontend/src/pages/DashboardPage.tsx

import React, { useEffect, useState } from 'react';
import { AlertPanel } from '../components/dashboard/AlertPanel';
import { StatsCard } from '../components/dashboard/StatsCard';
import { LiveMap } from '../components/dashboard/LiveMap';
import { EventTimeline } from '../components/dashboard/EventTimeline';
import { WaveformChart } from '../components/dashboard/WaveformChart';
```

```
import { dashboardAPI } from '../api/dashboard.api';
import { useRealTimeData } from '../hooks/useRealTimeData';

interface DashboardSummary {
  overview: {
    total_substations: number;
    active_feeders: number;
    total_events_today: number;
    active_events: number;
    resolved_events: number;
    model_accuracy: number;
  };
  recent_events: any[];
  alerts: {
    critical: number;
    high: number;
    medium: number;
    low: number;
  };
  performance: {
    avg_detection_time_ms: number;
    avg_response_time_minutes: number;
    false_positive_rate: number;
  };
}

export const DashboardPage: React.FC = () => {
  const [summary, setSummary] = useState<DashboardSummary | null>(null);
  const [loading, setLoading] = useState(true);

  // Real-time data subscription
```

```
const { data: realTimeEvents } = useRealTimeData('line_break_events');

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

useEffect(() => {
  // Refresh when new real-time event arrives
  if (realTimeEvents) {
    loadDashboardData();
  }
}, [realTimeEvents]);

const loadDashboardData = async () => {
  try {
    setLoading(true);
    const data = await dashboardAPI.getSummary();
    setSummary(data);
  } catch (error) {
    console.error('Failed to load dashboard data:', error);
  } finally {
    setLoading(false);
  }
};

if (loading || !summary) {
  return (
    <div className="flex items-center justify-center h-screen">
      <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-blue-600" />
    </div>
  );
```

```
}

return (
  <div className="p-6 space-y-6">
    {/* Header */}
    <div className="flex items-center justify-between">
      <div>
        <h1 className="text-3xl font-bold text-gray-900">
          Line Break Detection Dashboard
        </h1>
        <p className="text-gray-600 mt-1">
          Kerala State Electricity Board - Real-time Monitoring
        </p>
      </div>
      <div className="flex items-center gap-2">
        <span
          className={`inline-flex items-center px-3 py-1 rounded-full text-sm font-medium ${
            summary.overview.active_events > 0
              ? 'bg-red-100 text-red-800'
              : 'bg-green-100 text-green-800'
          }`}
        >
          <span className="w-2 h-2 mr-2 rounded-full bg-current animate-pulse" />
          {summary.overview.active_events > 0
            ? `${summary.overview.active_events} Active Alerts`
            : 'All Systems Normal'}
        </span>
      </div>
    </div>

    {/* Stats Cards */}
```

```
<div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-4 gap-6">
 <StatsCard
  title="Active Events"
  value={summary.overview.active_events}
  change={
   summary.overview.active_events > 0
    ? `${summary.overview.active_events} requiring attention`
    : 'No active events'
  }
  trend={summary.overview.active_events > 0 ? 'up' : 'neutral'}
  icon="alert-triangle"
  color="red"
 />
 <StatsCard
  title="Events Today"
  value={summary.overview.total_events_today}
  change={`${summary.overview.resolved_events} resolved`}
  trend="neutral"
  icon="activity"
  color="blue"
 />
 <StatsCard
  title="Model Accuracy"
  value={`${(summary.overview.model_accuracy * 100).toFixed(1)}%`}
  change={`${(summary.performance.false_positive_rate * 100).toFixed(2)}% false positive`}
  trend="up"
  icon="check-circle"
  color="green"
 />
 <StatsCard
  title="Avg Detection Time"
```

```
          value={`${summary.performance.avg_detection_time_ms}ms`}
          change={`${summary.performance.avg_response_time_minutes.toFixed(1)}min response`}
          trend="down"
          icon="zap"
          color="purple"
        />
      </div>

      {/* Main Content Grid */}
      <div className="grid grid-cols-1 lg:grid-cols-3 gap-6">
        {/* Alert Panel */}
        <div className="lg:col-span-1">
          <AlertPanel alerts={summary.alerts} events={summary.recent_events} />
        </div>

        {/* Map */}
        <div className="lg:col-span-2">
          <LiveMap events={summary.recent_events} />
        </div>
      </div>

      {/* Timeline and Charts */}
      <div className="grid grid-cols-1 lg:grid-cols-2 gap-6">
        <EventTimeline events={summary.recent_events} />
        <WaveformChart />
      </div>
    </div>
  );
};
```

4. Real-time Hook (Supabase Realtime)

typescript

```
// frontend/src/hooks/useRealTimeData.ts

import { useEffect, useState } from 'react';
import { supabase } from '../config/supabase';

export function useRealTimeData(table: string) {
  const [data, setData] = useState<any>(null);
  const [error, setError] = useState<Error | null>(null);

  useEffect(() => {
    // Subscribe to real-time changes
    const subscription = supabase
      .channel(`public:${table}`)
      .on(
        'postgres_changes',
        {
          event: '*',
          schema: 'public',
          table: table,
        },
        (payload) => {
          console.log('Real-time update:', payload);
          setData(payload.new);
        }
      )
      .subscribe();

    return () => {
      subscription.unsubscribe();
    };
```

```
  }, [table]);

  return { data, error };
}
```

📱 MOBILE APP (React Native + Expo)

App.tsx

typescript

```
// mobile/App.tsx

import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import { AuthProvider } from './src/context/AuthContext';

// Screens
import LoginScreen from './src/screens/LoginScreen';
import DashboardScreen from './src/screens/DashboardScreen';
import AlertsScreen from './src/screens/AlertsScreen';
import MapScreen from './src/screens/MapScreen';
import EventDetailScreen from './src/screens/EventDetailScreen';

const Stack = createNativeStackNavigator();

export default function App() {
  return (
    <AuthProvider>
      <NavigationContainer>
        <Stack.Navigator
          initialRouteName="Login"
          screenOptions={{
```

```
    headerStyle: {
      backgroundColor: '#dc2626',
    },
    headerTintColor: '#fff',
    headerTitleStyle: {
      fontWeight: 'bold',
    },
  }}
>
  <Stack.Screen
    name="Login"
    component={LoginScreen}
    options={{ headerShown: false }}
  />
  <Stack.Screen
    name="Dashboard"
    component={DashboardScreen}
    options={{ title: 'KSEBL Line Break Detection' }}
  />
  <Stack.Screen
    name="Alerts"
    component={AlertsScreen}
    options={{ title: 'Active Alerts' }}
  />
  <Stack.Screen
    name="Map"
    component={MapScreen}
    options={{ title: 'Event Map' }}
  />
  <Stack.Screen
    name="EventDetail"
```

```
          component={EventDetailScreen}
          options={{ title: 'Event Details' }}
        />
      </Stack.Navigator>
    </NavigationContainer>
  </AuthProvider>
 );
}
```

🚀 DEPLOYMENT (Vercel)

1. vercel.json

```
json
{
 "version": 2,
 "builds": [
  {
   "src": "backend/src/server.ts",
   "use": "@vercel/node"
  },
  {
   "src": "frontend/package.json",
   "use": "@vercel/static-build",
   "config": {
    "distDir": "dist"
   }
  }
 ],
 "routes": [
  {
   "src": "/api/(.*)",
   "dest": "backend/src/server.ts"
```

```
    },
    {
      "src": "/(.*)",
      "dest": "frontend/$1"
    }
  ],
  "env": {
    "NODE_ENV": "production"
  }
}
```

2. Environment Variables in Vercel

```
bash
# Go to Vercel Dashboard → Project → Settings → Environment Variables

# Add these variables:
SUPABASE_URL
SUPABASE_ANON_KEY
SUPABASE_SERVICE_KEY
JWT_SECRET
SMTP_HOST
SMTP_PORT
SMTP_USER
SMTP_PASSWORD
SMTP_FROM
TWILIO_ACCOUNT_SID (optional)
TWILIO_AUTH_TOKEN (optional)
TWILIO_PHONE_NUMBER (optional)
FRONTEND_URL
```

3. Deployment Steps

bash

```
# 1. Install Vercel CLI
npm i -g vercel

# 2. Login to Vercel
vercel login

# 3. Link project
vercel link

# 4. Deploy
vercel --prod

# Or use GitHub integration (recommended):
# - Push code to GitHub
# - Connect repo in Vercel dashboard
# - Auto-deploy on push to main branch
```

📊 DATA GENERATION SCRIPT (10,000 samples)

python

```
# scripts/generate_dataset.py

import numpy as np
import pandas as pd
from scipy import signal
import json
from datetime import datetime, timedelta
import random

class KeralaGridDataGenerator:
    """
```

```
Generate 10,000 realistic waveform samples for Kerala grid
"""
def __init__(self):
    self.sampling_rate = 10000  # 10 kHz
    self.feeders = self._generate_feeder_list()

def _generate_feeder_list(self):
    """Kerala KSEBL feeders"""
    districts = ['Trivandrum', 'Kollam', 'Pathanamthitta', 'Alappuzha',
                 'Kottayam', 'Idukki', 'Ernakulam', 'Thrissur', 'Palakkad',
                 'Malappuram', 'Kozhikode', 'Wayanad', 'Kannur', 'Kasaragod']

    feeders = []
    for district in districts:
        for i in range(1, 26):  # 25 feeders per district
            feeders.append({
                'id': f'{district[:3].upper()}-F{i:03d}',
                'name': f'{district} Feeder {i}',
                'district': district,
                'voltage': 11000,  # 11 kV
                'length_km': round(random.uniform(2, 15), 2)
            })
    return feeders

def generate_dataset(self, num_samples=10000):
    """Generate balanced dataset"""
    samples_per_class = num_samples // 4

    all_data = []

    print(f"Generating {num_samples} samples...")
```

```
    # Generate each class
    for class_name, generator_func in [
        ('NORMAL', self._generate_normal),
        ('LINE_BREAK', self._generate_line_break),
        ('SHORT_CIRCUIT', self._generate_short_circuit),
        ('OVERLOAD', self._generate_overload)
    ]:
        print(f"Generating {samples_per_class} {class_name} samples...")
        for i in range(samples_per_class):
            sample = generator_func()
            sample['label'] = class_name
            sample['sample_id'] = len(all_data) + 1
            all_data.append(sample)

            if (i + 1) % 100 == 0:
                print(f"  Progress: {i+1}/{samples_per_class}")

    return all_data

def _generate_normal(self):
    feeder = random.choice(self.feeders)
    base_current = random.uniform(30, 60)
    base_voltage = 230

    t = np.linspace(0, 4, 4 * self.sampling_rate)

    # 3-phase currents with realistic variations
    load_var = np.random.normal(1.0, 0.05, len(t))
    current_r = base_current * load_var * np.sin(2*np.pi*50*t)
    current_y = base_current * load_var * np.sin(2*np.pi*50*t - 2*np.pi/3)
```

```
    current_b = base_current * load_var * np.sin(2*np.pi*50*t + 2*np.pi/3)

    # Add harmonics
    for h in [3, 5, 7]:
        harm_amp = base_current * 0.03 / h
        current_r += harm_amp * np.sin(2*np.pi*50*h*t)
        current_y += harm_amp * np.sin(2*np.pi*50*h*t - 2*np.pi/3)
        current_b += harm_amp * np.sin(2*np.pi*50*h*t + 2*np.pi/3)

    # Voltages
    voltage_r = base_voltage * np.sin(2*np.pi*50*t)
    voltage_y = base_voltage * np.sin(2*np.pi*50*t - 2*np.pi/3)
    voltage_b = base_voltage * np.sin(2*np.pi*50*t + 2*np.pi/3)

    return {
        'feeder_id': feeder['id'],
        'feeder_name': feeder['name'],
        'timestamp': (datetime.now() - timedelta(days=random.randint(0, 365))).isoformat(),
        'current_r': current_r.tolist()[::100],  # Downsample for storage
        'current_y': current_y.tolist()[::100],
        'current_b': current_b.tolist()[::100],
        'voltage_r': voltage_r.tolist()[::100],
        'voltage_y': voltage_y.tolist()[::100],
        'voltage_b': voltage_b.tolist()[::100],
    }

def _generate_line_break(self):
    sample = self._generate_normal()

    # Inject line break at random point
    break_point = random.randint(len(sample['current_r']) // 3, 2 * len(sample['current_r']) // 3)
```

```
        drop_factor = random.uniform(0.2, 0.5)

        for phase in ['current_r', 'current_y', 'current_b']:
            sample[phase][break_point:] = [x * drop_factor for x in sample[phase][break_point:]]

        for phase in ['voltage_r', 'voltage_y', 'voltage_b']:
            sample[phase][break_point:] = [x * 0.85 for x in sample[phase][break_point:]]

        sample['break_location_km'] = round(random.uniform(0.5, sample.get('feeder_length_km', 10)),
2)

        return sample

    def _generate_short_circuit(self):
        sample = self._generate_normal()
        fault_point = random.randint(len(sample['current_r']) // 3, 2 * len(sample['current_r']) // 3)

        for phase in ['current_r', 'current_y', 'current_b']:
            sample[phase][fault_point:] = [x * 8 for x in sample[phase][fault_point:]]

        for phase in ['voltage_r', 'voltage_y', 'voltage_b']:
            sample[phase][fault_point:] = [x * 0.1 for x in sample[phase][fault_point:]]

        return sample

    def _generate_overload(self):
        sample = self._generate_normal()
        overload_point = len(sample['current_r']) // 4

        for i in range(overload_point, len(sample['current_r'])):
            factor = 1.0 + (i - overload_point) / (len(sample['current_r']) - overload_point) * 0.5
```

```
            sample['current_r'][i] *= factor
            sample['current_y'][i] *= factor
            sample['current_b'][i] *= factor

        return sample

# Generate and save
if __name__ == '__main__':
    generator = KeralaGridDataGenerator()
    dataset = generator.generate_dataset(num_samples=10000)

    # Save to JSON
    with open('kerala_grid_dataset_10k.json', 'w') as f:
        json.dump(dataset, f)

    print(f"\n✅ Generated {len(dataset)} samples")
    print(f"Dataset saved to: kerala_grid_dataset_10k.json")
```

📝 TO-DO LIST (Execute One at a Time)

```
markdown
# MVP Development Checklist

## Phase 1: Setup & Configuration (Day 1-2)
- [ ] 1.1 Create Supabase project and get credentials
- [ ] 1.2 Execute SQL schema in Supabase SQL Editor
- [ ] 1.3 Create `.env` file with all credentials
- [ ] 1.4 Initialize Git repository
- [ ] 1.5 Create project folder structure
- [ ] 1.6 Install backend dependencies (`npm install`)
- [ ] 1.7 Install frontend dependencies (`npm install`)
- [ ] 1.8 Create `.cursorrules` file
```

- [ ] 1.9 Test Supabase connection from backend
- [ ] 1.10 Test SMTP email configuration

Phase 2: Backend Core (Day 3-5)
- [ ] 2.1 Implement Supabase config (`backend/src/config/supabase.ts`)
- [ ] 2.2 Implement environment validation (`backend/src/config/env.ts`)
- [ ] 2.3 Create auth middleware (`backend/src/middleware/auth.middleware.ts`)
- [ ] 2.4 Create error handling middleware (`backend/src/middleware/error.middleware.ts`)
- [ ] 2.5 Implement API response utilities (`backend/src/utils/ApiResponse.ts`)
- [ ] 2.6 Create user model and types (`backend/src/models/User.ts`)
- [ ] 2.7 Implement auth controller (register, login, logout)
- [ ] 2.8 Create auth routes (`backend/src/routes/auth.routes.ts`)
- [ ] 2.9 Test authentication flow with Postman/Thunder Client
- [ ] 2.10 Create logger utility (`backend/src/utils/logger.ts`)

Phase 3: Data Generation (Day 6-7)
- [ ] 3.1 Install Python dependencies for data generation
- [ ] 3.2 Run `generate_dataset.py` to create 10,000 samples
- [ ] 3.3 Verify generated dataset structure
- [ ] 3.4 Create database seed script (`scripts/seed-database.ts`)
- [ ] 3.5 Seed substations table (Kerala districts)
- [ ] 3.6 Seed feeders table (25 per district)
- [ ] 3.7 Seed users table (admin, operators, field crew)
- [ ] 3.8 Upload sample waveform data to Supabase
- [ ] 3.9 Verify data in Supabase dashboard
- [ ] 3.10 Create backup of seeded data

Phase 4: ML Implementation (Day 8-10)
- [ ] 4.1 Implement feature extractor (`backend/src/services/featureExtractor.ts`)
- [ ] 4.2 Test feature extraction on sample data
- [ ] 4.3 Train ML model using Python (or use pre-trained)

- [ ] 4.4 Export model to TensorFlow.js format
- [ ] 4.5 Implement ML service (`backend/src/services/mlService.ts`)
- [ ] 4.6 Test ML inference with sample waveforms
- [ ] 4.7 Optimize model loading and inference time
- [ ] 4.8 Create model performance metrics endpoint
- [ ] 4.9 Implement batch prediction functionality
- [ ] 4.10 Document model accuracy and performance

Phase 5: Event Management (Day 11-13)
- [ ] 5.1 Create event model and types
- [ ] 5.2 Implement event controller (CRUD operations)
- [ ] 5.3 Create event routes
- [ ] 5.4 Implement waveform controller (upload, analyze)
- [ ] 5.5 Create waveform routes
- [ ] 5.6 Implement real-time event creation
- [ ] 5.7 Test event creation and status updates
- [ ] 5.8 Implement event assignment to field crew
- [ ] 5.9 Implement event resolution workflow
- [ ] 5.10 Test complete event lifecycle

Phase 6: Alert System (Day 14-15)
- [ ] 6.1 Implement alert service (`backend/src/services/alertService.ts`)
- [ ] 6.2 Configure email templates for alerts
- [ ] 6.3 Test email sending functionality
- [ ] 6.4 Configure SMS service (Twilio)
- [ ] 6.5 Test SMS sending functionality
- [ ] 6.6 Implement alert routing logic (who gets notified)
- [ ] 6.7 Create alerts table queries
- [ ] 6.8 Implement alert status tracking
- [ ] 6.9 Test alert delivery for different event severities
- [ ] 6.10 Implement alert retry mechanism for failures

Phase 7: Dashboard & Analytics (Day 16-18)

- [ ] 7.1 Create dashboard controller
- [ ] 7.2 Implement summary statistics endpoint
- [ ] 7.3 Implement recent events endpoint
- [ ] 7.4 Implement map data endpoint (GeoJSON)
- [ ] 7.5 Create analytics controller
- [ ] 7.6 Implement trends analysis endpoint
- [ ] 7.7 Implement performance metrics endpoint
- [ ] 7.8 Test all dashboard endpoints
- [ ] 7.9 Optimize queries for dashboard performance
- [ ] 7.10 Create dashboard routes

Phase 8: Frontend Core (Day 19-21)

- [ ] 8.1 Setup Vite React TypeScript project
- [ ] 8.2 Install dependencies (Tailwind, React Router, etc.)
- [ ] 8.3 Configure Tailwind CSS
- [ ] 8.4 Create API client (`frontend/src/api/client.ts`)
- [ ] 8.5 Implement auth context and hooks
- [ ] 8.6 Create login page
- [ ] 8.7 Create protected route wrapper
- [ ] 8.8 Implement auth API calls
- [ ] 8.9 Test login/logout flow
- [ ] 8.10 Create layout components (Header, Sidebar)

Phase 9: Dashboard UI (Day 22-24)

- [ ] 9.1 Create stats card component
- [ ] 9.2 Create alert panel component
- [ ] 9.3 Implement dashboard page layout
- [ ] 9.4 Integrate dashboard API
- [ ] 9.5 Create live map component (Leaflet/Google Maps)

- [ ] 9.6 Plot events on map with markers
- [ ] 9.7 Create event timeline component
- [ ] 9.8 Create waveform chart component (Recharts)
- [ ] 9.9 Implement real-time updates (Supabase Realtime)
- [ ] 9.10 Test dashboard responsiveness

Phase 10: Event Management UI (Day 25-26)
- [ ] 10.1 Create events list page
- [ ] 10.2 Implement event filtering (status, date)
- [ ] 10.3 Create event detail page
- [ ] 10.4 Implement event status update UI
- [ ] 10.5 Create event assignment modal
- [ ] 10.6 Implement event resolution form
- [ ] 10.7 Add event action buttons (acknowledge, assign, resolve)
- [ ] 10.8 Implement event API integration
- [ ] 10.9 Test event management workflow
- [ ] 10.10 Add loading states and error handling

Phase 11: Additional Features (Day 27-28)
- [ ] 11.1 Create substations list page
- [ ] 11.2 Create feeders list page
- [ ] 11.3 Create users management page (admin only)
- [ ] 11.4 Implement settings page
- [ ] 11.5 Add notification system (toast messages)
- [ ] 11.6 Implement search functionality
- [ ] 11.7 Add export functionality (CSV/PDF)
- [ ] 11.8 Create analytics page with charts
- [ ] 11.9 Implement dark mode toggle
- [ ] 11.10 Add accessibility features (ARIA labels)

Phase 12: Mobile App (Day 29-30)

- [ ] 12.1 Initialize Expo React Native project
- [ ] 12.2 Install dependencies (React Navigation, etc.)
- [ ] 12.3 Create navigation structure
- [ ] 12.4 Implement login screen
- [ ] 12.5 Create dashboard screen
- [ ] 12.6 Create alerts list screen
- [ ] 12.7 Create map screen
- [ ] 12.8 Create event detail screen
- [ ] 12.9 Implement push notifications
- [ ] 12.10 Test on Android and iOS

Phase 13: Testing (Day 31-32)
- [ ] 13.1 Write unit tests for auth controller
- [ ] 13.2 Write unit tests for event controller
- [ ] 13.3 Write unit tests for ML service
- [ ] 13.4 Write integration tests for API endpoints
- [ ] 13.5 Test error handling scenarios
- [ ] 13.6 Test authentication and authorization
- [ ] 13.7 Perform load testing (Artillery/k6)
- [ ] 13.8 Test frontend components (React Testing Library)
- [ ] 13.9 Perform E2E testing (Cypress/Playwright)
- [ ] 13.10 Fix identified bugs

Phase 14: Deployment Preparation (Day 33-34)
- [ ] 14.1 Create production environment variables
- [ ] 14.2 Configure vercel.json
- [ ] 14.3 Optimize frontend bundle size
- [ ] 14.4 Optimize backend performance
- [ ] 14.5 Setup database indexes
- [ ] 14.6 Configure CORS for production
- [ ] 14.7 Setup logging and monitoring

- [ ] 14.8 Create deployment documentation
- [ ] 14.9 Setup CI/CD pipeline (GitHub Actions)
- [ ] 14.10 Create environment variable checklist

Phase 15: Deployment (Day 35)
- [ ] 15.1 Deploy backend to Vercel
- [ ] 15.2 Configure Vercel environment variables
- [ ] 15.3 Deploy frontend to Vercel
- [ ] 15.4 Test production deployment
- [ ] 15.5 Setup custom domain (if available)
- [ ] 15.6 Configure SSL certificate
- [ ] 15.7 Test all API endpoints in production
- [ ] 15.8 Test frontend in production
- [ ] 15.9 Monitor error logs
- [ ] 15.10 Create rollback plan

Phase 16: Documentation (Day 36)
- [ ] 16.1 Write comprehensive README.md
- [ ] 16.2 Document API endpoints (Swagger/OpenAPI)
- [ ] 16.3 Create architecture diagram
- [ ] 16.4 Write deployment guide
- [ ] 16.5 Create user manual
- [ ] 16.6 Document database schema
- [ ] 16.7 Write ML model documentation
- [ ] 16.8 Create troubleshooting guide
- [ ] 16.9 Document environment variables
- [ ] 16.10 Create video demo

Phase 17: Hackathon Preparation (Day 37-38)
- [ ] 17.1 Prepare presentation slides
- [ ] 17.2 Create demo script

- [ ] 17.3 Record demo video
- [ ] 17.4 Prepare pitch (problem, solution, impact)
- [ ] 17.5 Create ROI analysis
- [ ] 17.6 Prepare technical architecture slides
- [ ] 17.7 Create live demo scenarios
- [ ] 17.8 Prepare Q&A responses
- [ ] 17.9 Test demo on different devices
- [ ] 17.10 Finalize presentation

Phase 18: Final Polish (Day 39-40)
- [ ] 18.1 Fix UI/UX issues
- [ ] 18.2 Optimize loading times
- [ ] 18.3 Add animations and transitions
- [ ] 18.4 Improve error messages
- [ ] 18.5 Add help tooltips
- [ ] 18.6 Improve mobile responsiveness
- [ ] 18.7 Test on multiple browsers
- [ ] 18.8 Validate all forms
- [ ] 18.9 Check accessibility compliance
- [ ] 18.10 Final security audit

🔧 UTILITY FILES

ApiError Class

typescript

```
// backend/src/utils/ApiError.ts

export class ApiError extends Error {
  statusCode: number;
  code: string;
  isOperational: boolean;
```

```
  constructor(statusCode: number, message: string, code?: string) {
    super(message);
    this.statusCode = statusCode;
    this.code = code || 'ERROR';
    this.isOperational = true;

    Error.captureStackTrace(this, this.constructor);
  }
}
```

ApiResponse Class

typescript

```
// backend/src/utils/ApiResponse.ts

export class ApiResponse<T = any> {
  success: boolean;
  data?: T;
  message?: string;
  error?: {
    code: string;
    message: string;
  };

  constructor(success: boolean, data?: T, message?: string) {
    this.success = success;
    if (data !== undefined) this.data = data;
    if (message) this.message = message;
  }
}
```

Error Middleware

typescript

```
// backend/src/middleware/error.middleware.ts

import { Request, Response, NextFunction } from 'express';
import { ApiError } from '../utils/ApiError';
import { config } from '../config/env';

export const errorHandler = (
  err: Error | ApiError,
  req: Request,
  res: Response,
  next: NextFunction
) => {
  let statusCode = 500;
  let message = 'Internal Server Error';
  let code = 'INTERNAL_ERROR';

  if (err instanceof ApiError) {
    statusCode = err.statusCode;
    message = err.message;
    code = err.code;
  }

  // Log error
  console.error('Error:', {
    code,
    message,
    statusCode,
    path: req.path,
    method: req.method,
    stack: config.env === 'development' ? err.stack : undefined,
```

```
  });

  res.status(statusCode).json({
    success: false,
    error: {
      code,
      message,
      ...(config.env === 'development' && { stack: err.stack }),
    },
  });
};
```

Frontend Environment

typescript

```
// frontend/.env.example

VITE_API_URL=http://localhost:5000/api/v1
VITE_SUPABASE_URL=https://your-project.supabase.co
VITE_SUPABASE_ANON_KEY=your-anon-key
VITE_GOOGLE_MAPS_API_KEY=your-google-maps-key
```

Package.json Scripts

json

```
// backend/package.json

{
  "name": "kerala-line-break-backend",
  "version": "1.0.0",
  "scripts": {
    "dev": "nodemon src/server.ts",
    "build": "tsc",
```

```
  "start": "node dist/server.js",
  "test": "jest",
  "lint": "eslint . --ext .ts",
  "seed": "ts-node scripts/seed-database.ts"
},
"dependencies": {
  "@supabase/supabase-js": "^2.38.4",
  "@tensorflow/tfjs-node": "^4.13.0",
  "express": "^4.18.2",
  "bcrypt": "^5.1.1",
  "jsonwebtoken": "^9.0.2",
  "dotenv": "^16.3.1",
  "zod": "^3.22.4",
  "nodemailer": "^6.9.7",
  "twilio": "^4.19.0",
  "cors": "^2.8.5",
  "helmet": "^7.1.0",
  "compression": "^1.7.4",
  "morgan": "^1.10.0",
  "express-rate-limit": "^7.1.5"
},
"devDependencies": {
  "@types/express": "^4.17.21",
  "@types/node": "^20.10.4",
  "@types/bcrypt": "^5.0.2",
  "@types/jsonwebtoken": "^9.0.5",
  "@types/nodemailer": "^6.4.14",
  "@types/cors": "^2.8.17",
  "@types/compression": "^1.7.5",
  "@types/morgan": "^1.9.9",
  "typescript": "^5.3.3",
```

```
    "nodemon": "^3.0.2",
    "ts-node": "^10.9.2",
    "jest": "^29.7.0",
    "@types/jest": "^29.5.11",
    "eslint": "^8.55.0",
    "@typescript-eslint/parser": "^6.14.0",
    "@typescript-eslint/eslint-plugin": "^6.14.0"
  }
}
```

json

```
// frontend/package.json

{
  "name": "kerala-line-break-frontend",
  "version": "1.0.0",
  "scripts": {
    "dev": "vite",
    "build": "tsc && vite build",
    "preview": "vite preview",
    "lint": "eslint . --ext ts,tsx",
    "test": "jest"
  },
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.20.1",
    "@supabase/supabase-js": "^2.38.4",
    "axios": "^1.6.2",
    "recharts": "^2.10.3",
    "leaflet": "^1.9.4",
```

```
    "react-leaflet": "^4.2.1",
    "lucide-react": "^0.294.0",
    "clsx": "^2.0.0",
    "tailwind-merge": "^2.1.0"
  },
  "devDependencies": {
    "@types/react": "^18.2.43",
    "@types/react-dom": "^18.2.17",
    "@types/leaflet": "^1.9.8",
    "@vitejs/plugin-react": "^4.2.1",
    "vite": "^5.0.7",
    "typescript": "^5.3.3",
    "tailwindcss": "^3.3.6",
    "autoprefixer": "^10.4.16",
    "postcss": "^8.4.32",
    "eslint": "^8.55.0",
    "@typescript-eslint/parser": "^6.14.0",
    "@typescript-eslint/eslint-plugin": "^6.14.0"
  }
}
```

🎯 CURSOR AI INTEGRATION INSTRUCTIONS

How to Use with Cursor

```
bash
# 1. Open project in Cursor
cursor .

# 2. Load .cursorrules file (automatically detected)

# 3. Use Cursor commands:
# - Ask Cursor to implement specific task from TO-DO list
```

- Reference .cursorrules for coding standards

- Use CMD+K (Mac) or CTRL+K (Windows) for inline edits

Example prompts:

"Implement the auth middleware following the .cursorrules"

"Create the event controller with proper error handling"

"Add TypeScript types for the dashboard API response"

"Refactor this code to follow clean code principles"

Cursor Workflow

markdown

1. Open TO-DO list

2. Pick ONE task (e.g., "2.3 Create auth middleware")

3. Ask Cursor: "Implement task 2.3 from the TO-DO list following .cursorrules"

4. Review generated code

5. Test the implementation

6. Commit to Git

7. Move to next task

8. Repeat

📊 SUCCESS METRICS

markdown

MVP Success Criteria

Technical Metrics

✅ Detection Accuracy: >95%

✅ Detection Speed: <200ms

✅ API Response Time: <500ms

✅ Uptime: >99.5%

✅ False Positive Rate: <2%

Functional Metrics

✅ User Authentication: Working

✅ Event Detection: Automated

✅ Alert Delivery: <30 seconds

✅ Dashboard: Real-time updates

✅ Mobile App: iOS + Android support

Business Metrics

✅ Demo-ready: Yes

✅ Scalable: 500+ substations

✅ Cost-effective: <₹50L deployment

✅ ROI: <6 months payback

✅ Lives Saved: Projected 50-100/year

🚀 FINAL DEPLOYMENT CHECKLIST

markdown

Pre-Deployment Checklist

Security

- [ ] All secrets in environment variables
- [ ] No hardcoded credentials
- [ ] HTTPS enabled
- [ ] CORS configured correctly
- [ ] Rate limiting enabled
- [ ] Input validation on all endpoints
- [ ] SQL injection protection
- [ ] XSS protection

Performance

- [ ] Database indexes created

- [ ] API responses optimized
- [ ] Frontend bundle optimized
- [ ] Images optimized
- [ ] Caching implemented
- [ ] CDN configured (if needed)

Monitoring
- [ ] Error logging configured
- [ ] Performance monitoring setup
- [ ] Uptime monitoring
- [ ] Alert notifications working

Documentation
- [ ] README.md complete
- [ ] API documentation ready
- [ ] Deployment guide written
- [ ] User manual created

Testing
- [ ] All unit tests passing
- [ ] Integration tests passing
- [ ] E2E tests passing
- [ ] Load testing completed
- [ ] Security audit done

Backup
- [ ] Database backup strategy
- [ ] Code backup (Git)
- [ ] Environment variables documented
- [ ] Rollback plan ready

📞 SUPPORT & RESOURCES

markdown

Resources

Supabase

- Docs: https://supabase.com/docs

- Dashboard: https://app.supabase.com

Vercel

- Docs: https://vercel.com/docs

- Dashboard: https://vercel.com/dashboard

Tech Stack Docs

- Node.js: https://nodejs.org/docs

- TypeScript: https://www.typescriptlang.org/docs

- React: https://react.dev

- TensorFlow.js: https://www.tensorflow.org/js

Kerala KSEBL

- Official Site: https://www.kseb.in

- Contact: For real data access

🎉 YOU'RE READY TO BUILD!

This MVP prompt contains:

✅ Complete system architecture

✅ Database schema with 10+ tables

✅ Full backend in Node.js + TypeScript

✅ Full frontend in React + TypeScript

✅ Mobile app in React Native

✅ ML model integration

- ✅ Real-time alerts (Email + SMS)
- ✅ Deployment on Vercel
- ✅ 10,000 sample dataset generator
- ✅ .cursorrules for code standards
- ✅ Step-by-step TO-DO list
- ✅ API documentation
- ✅ Testing strategy

Start with Phase 1, Task 1.1 and execute ONE TASK AT A TIME!