Complaint Management System - Complete Implementation

Project Structure

```
complaint-management-system/
├── backend/
│   ├── app/
│   │   ├── __init__.py
│   │   ├── main.py
│   │   ├── models/
│   │   │   ├── __init__.py
│   │   │   └── complaint.py
│   │   ├── api/
│   │   │   ├── __init__.py
│   │   │   └── complaints.py
│   │   ├── services/
│   │   │   ├── __init__.py
│   │   │   └── complaint_service.py
│   │   ├── ai/
│   │   │   ├── __init__.py
│   │   │   └── classifier.py
│   │   └── config/
│   │       ├── __init__.py
│   │       └── config.py
│   ├── config/
│   │   ├── categories.yaml
│   │   ├── priorities.yaml
│   │   └── features.yaml
│   ├── requirements.txt
│   └── run.py
├── frontend/
│   ├── public/
│   │   └── index.html
│   ├── src/
│   │   ├── components/
│   │   │   ├── ComplaintForm.jsx
│   │   │   ├── ComplaintList.jsx
│   │   │   └── ComplaintCard.jsx
│   │   ├── services/
│   │   │   └── api.js
│   │   ├── store/
│   │   │   └── ComplaintStore.jsx
│   │   ├── App.jsx
│   │   └── index.js
│   ├── package.json
```

```
│   └── webpack.config.js
└── docker-compose.yml
```

Backend Implementation

1. Backend Dependencies (requirements.txt)

```
Flask==2.3.3
Flask-SQLAlchemy==3.0.5
Flask-CORS==4.0.0
PyYAML==6.0.1
scikit-learn==1.3.0
python-dotenv==1.0.0
gunicorn==21.2.0
```

2. Configuration Files

config/categories.yaml

yaml

```yaml
categories:
  - id: "water_supply"
    name: "Water Supply Issues"
    keywords: ["water", "tap", "supply", "leak"]
    department: "water_dept"
    auto_priority: "medium"
  - id: "road_maintenance"
    name: "Road Maintenance"
    keywords: ["road", "pothole", "street", "repair"]
    department: "public_works"
    auto_priority: "high"
  - id: "waste_management"
    name: "Waste Management"
    keywords: ["garbage", "trash", "waste", "collection"]
    department: "sanitation"
    auto_priority: "medium"
  - id: "electrical"
    name: "Electrical Issues"
    keywords: ["power", "electricity", "outage", "pole"]
    department: "electrical"
    auto_priority: "high"
```

config/priorities.yaml

yaml

```yaml
priorities:
  - id: "low"
    name: "Low Priority"
    sla_hours: 168  # 7 days
    color: "#28a745"
  - id: "medium"
    name: "Medium Priority"
    sla_hours: 72   # 3 days
    color: "#ffc107"
  - id: "high"
    name: "High Priority"
    sla_hours: 24   # 1 day
    color: "#dc3545"
```

config/features.yaml

```yaml
features:
  voice_input: false
  ai_classification: true
  auto_assignment: true
  email_notifications: false
  sms_notifications: false
  advanced_analytics: false
  bulk_operations: false
```

3. Backend Models (app/models/complaint.py)

```python
```

```python
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime
import json

db = SQLAlchemy()

class ComplaintCategory(db.Model):
    __tablename__ = 'complaint_categories'

    id = db.Column(db.String(50), primary_key=True)
    name = db.Column(db.String(100), nullable=False)
    keywords = db.Column(db.JSON)
    department_id = db.Column(db.String(50))
    default_priority = db.Column(db.String(20))
    is_active = db.Column(db.Boolean, default=True)

class ComplaintPriority(db.Model):
    __tablename__ = 'complaint_priorities'

    id = db.Column(db.String(20), primary_key=True)
    name = db.Column(db.String(50), nullable=False)
    sla_hours = db.Column(db.Integer)
    color_code = db.Column(db.String(7))
    order_index = db.Column(db.Integer)

class Complaint(db.Model):
    __tablename__ = 'complaints'

    id = db.Column(db.String(36), primary_key=True)
    user_id = db.Column(db.String(36))
    title = db.Column(db.String(200))
    description = db.Column(db.Text)
    category_id = db.Column(db.String(50))
    priority_id = db.Column(db.String(20))
    metadata = db.Column(db.JSON)
    status = db.Column(db.String(20), default='open')
    assigned_to = db.Column(db.String(36))
    created_at = db.Column(db.DateTime, default=datetime.utcnow)
    updated_at = db.Column(db.DateTime, default=datetime.utcnow)

    def to_dict(self):
        return {
            'id': self.id,
            'user_id': self.user_id,
            'title': self.title,
            'description': self.description,
```

```
        'category_id': self.category_id,
        'priority_id': self.priority_id,
        'status': self.status,
        'assigned_to': self.assigned_to,
        'created_at': self.created_at.isoformat() if self.created_at else None,
        'updated_at': self.updated_at.isoformat() if self.updated_at else None,
        'metadata': self.metadata or {}
    }
```

4. AI Classifier (app/ai/classifier.py)

python

```python
import yaml
import re
from typing import Dict, List

class SimpleKeywordClassifier:
    def __init__(self, categories_path: str):
        with open(categories_path, 'r') as f:
            data = yaml.safe_load(f)
        self.categories = data['categories']

    def classify(self, text: str) -> Dict:
        text_lower = text.lower()
        best_category = None
        max_matches = 0

        for category in self.categories:
            matches = 0
            for keyword in category['keywords']:
                if keyword.lower() in text_lower:
                    matches += 1

            if matches > max_matches:
                max_matches = matches
                best_category = category

        if best_category:
            return {
                'category': best_category['id'],
                'priority': best_category['auto_priority'],
                'confidence': min(max_matches / len(best_category['keywords']), 1.0)
            }

        return {
            'category': 'general',
            'priority': 'low',
            'confidence': 0.0
        }

class PriorityAnalyzer:
    def __init__(self, priorities_path: str):
        with open(priorities_path, 'r') as f:
            data = yaml.safe_load(f)
        self.priorities = {p['id']: p for p in data['priorities']}

    def analyze_priority(self, text: str, category: str) -> str:
        text_lower = text.lower()
```

```
    urgent_keywords = ['urgent', 'emergency', 'critical', 'immediate', 'danger']

    if any(keyword in text_lower for keyword in urgent_keywords):
        return 'high'

    return 'medium'  # Default fallback
```

5. Service Layer (app/services/complaint_service.py)

python

```python
import uuid
from datetime import datetime
from app.models.complaint import db, Complaint, ComplaintCategory, ComplaintPriority
from app.ai.classifier import SimpleKeywordClassifier, PriorityAnalyzer

class ComplaintService:
    def __init__(self):
        self.classifier = SimpleKeywordClassifier('config/categories.yaml')
        self.priority_analyzer = PriorityAnalyzer('config/priorities.yaml')

    def create_complaint(self, complaint_data: dict) -> dict:
        try:
            # Generate unique ID
            complaint_id = str(uuid.uuid4())

            # AI Classification
            classification = self.classifier.classify(
                complaint_data.get('description', '') + ' ' + complaint_data.get('title', '')
            )

            # Create complaint object
            complaint = Complaint(
                id=complaint_id,
                user_id=complaint_data.get('user_id', 'anonymous'),
                title=complaint_data['title'],
                description=complaint_data['description'],
                category_id=classification.get('category', 'general'),
                priority_id=classification.get('priority', 'low'),
                status='open',
                metadata={'ai_classified': True, 'confidence': classification.get('confidence', 0.0)}
            )

            db.session.add(complaint)
            db.session.commit()

            return {'status': 'success', 'complaint': complaint.to_dict()}

        except Exception as e:
            db.session.rollback()
            return {'status': 'error', 'message': str(e)}

    def get_complaints(self, filters: dict = None) -> List[dict]:
        query = Complaint.query

        if filters:
            if filters.get('status'):
```

```python
            query = query.filter(Complaint.status == filters['status'])
        if filters.get('category'):
            query = query.filter(Complaint.category_id == filters['category'])
        if filters.get('priority'):
            query = query.filter(Complaint.priority_id == filters['priority'])

    complaints = query.order_by(Complaint.created_at.desc()).all()
    return [complaint.to_dict() for complaint in complaints]

def update_complaint_status(self, complaint_id: str, status: str) -> dict:
    try:
        complaint = Complaint.query.get(complaint_id)
        if not complaint:
            return {'status': 'error', 'message': 'Complaint not found'}

        complaint.status = status
        complaint.updated_at = datetime.utcnow()
        db.session.commit()

        return {'status': 'success', 'complaint': complaint.to_dict()}

    except Exception as e:
        db.session.rollback()
        return {'status': 'error', 'message': str(e)}

def get_categories(self) -> List[dict]:
    categories = ComplaintCategory.query.filter_by(is_active=True).all()
    return [{'id': c.id, 'name': c.name} for c in categories]

def get_priorities(self) -> List[dict]:
    priorities = ComplaintPriority.query.order_by(ComplaintPriority.order_index).all()
    return [{'id': p.id, 'name': p.name, 'color': p.color_code} for p in priorities]
```

6. API Routes (app/api/complaints.py)

python

```python
from flask import Blueprint, request, jsonify
from app.services.complaint_service import ComplaintService

complaints_bp = Blueprint('complaints', __name__, url_prefix='/api/v1')
complaint_service = ComplaintService()

@complaints_bp.route('/complaints', methods=['POST'])
def create_complaint():
    data = request.get_json()

    if not data or not data.get('title') or not data.get('description'):
        return jsonify({'error': 'Title and description are required'}), 400

    result = complaint_service.create_complaint(data)

    if result['status'] == 'success':
        return jsonify(result), 201
    else:
        return jsonify(result), 400

@complaints_bp.route('/complaints', methods=['GET'])
def get_complaints():
    filters = {
        'status': request.args.get('status'),
        'category': request.args.get('category'),
        'priority': request.args.get('priority')
    }
    # Remove None values
    filters = {k: v for k, v in filters.items() if v is not None}

    complaints = complaint_service.get_complaints(filters)
    return jsonify({'complaints': complaints})

@complaints_bp.route('/complaints/<complaint_id>', methods=['PUT'])
def update_complaint(complaint_id):
    data = request.get_json()

    if 'status' in data:
        result = complaint_service.update_complaint_status(complaint_id, data['status'])
        if result['status'] == 'success':
            return jsonify(result)
        else:
            return jsonify(result), 400

    return jsonify({'error': 'No valid updates provided'}), 400
```

```python
@complaints_bp.route('/categories', methods=['GET'])
def get_categories():
    categories = complaint_service.get_categories()
    return jsonify({'categories': categories})

@complaints_bp.route('/priorities', methods=['GET'])
def get_priorities():
    priorities = complaint_service.get_priorities()
    return jsonify({'priorities': priorities})
```

7. Configuration Manager (app/config/config.py)

```python
import os
import yaml
from dataclasses import dataclass

@dataclass
class Config:
    DATABASE_URL: str
    AI_CLASSIFIER_TYPE: str
    ENABLE_AI_CLASSIFICATION: bool
    ENABLE_AUTO_ASSIGNMENT: bool

    @classmethod
    def load_from_env(cls):
        return cls(
            DATABASE_URL=os.getenv('DATABASE_URL', 'sqlite:///complaints.db'),
            AI_CLASSIFIER_TYPE=os.getenv('AI_CLASSIFIER', 'keyword'),
            ENABLE_AI_CLASSIFICATION=os.getenv('ENABLE_AI_CLASSIFICATION', 'true').lower() == 'true',
            ENABLE_AUTO_ASSIGNMENT=os.getenv('ENABLE_AUTO_ASSIGNMENT', 'true').lower() == 'true'
        )

def load_yaml_config(file_path: str):
    with open(file_path, 'r') as f:
        return yaml.safe_load(f)
```

8. Main Application (app/main.py)

```python
```

```python
from flask import Flask
from flask_cors import CORS
from app.models.complaint import db, ComplaintCategory, ComplaintPriority
from app.api.complaints import complaints_bp
from app.config.config import Config, load_yaml_config
import yaml

def create_app():
    app = Flask(__name__)
    CORS(app)

    # Load configuration
    config = Config.load_from_env()
    app.config['SQLALCHEMY_DATABASE_URI'] = config.DATABASE_URL
    app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

    # Initialize database
    db.init_app(app)

    # Register blueprints
    app.register_blueprint(complaints_bp)

    with app.app_context():
        db.create_all()
        initialize_data()

    return app

def initialize_data():
    """Initialize categories and priorities from YAML files"""
    # Load categories
    if ComplaintCategory.query.count() == 0:
        with open('config/categories.yaml', 'r') as f:
            categories_data = yaml.safe_load(f)

        for cat_data in categories_data['categories']:
            category = ComplaintCategory(
                id=cat_data['id'],
                name=cat_data['name'],
                keywords=cat_data['keywords'],
                department_id=cat_data['department'],
                default_priority=cat_data['auto_priority']
            )
            db.session.add(category)

    # Load priorities
```

```
    if ComplaintPriority.query.count() == 0:
        with open('config/priorities.yaml', 'r') as f:
            priorities_data = yaml.safe_load(f)

        for i, pri_data in enumerate(priorities_data['priorities']):
            priority = ComplaintPriority(
                id=pri_data['id'],
                name=pri_data['name'],
                sla_hours=pri_data['sla_hours'],
                color_code=pri_data['color'],
                order_index=i
            )
            db.session.add(priority)

        db.session.commit()

if __name__ == '__main__':
    app = create_app()
    app.run(debug=True, host='0.0.0.0', port=5000)
```

9. Application Runner (run.py)

```
python

from app.main import create_app

app = create_app()

if __name__ == '__main__':
    app.run(debug=True, host='0.0.0.0', port=5000)
```

Frontend Implementation

1. Frontend Dependencies (package.json)

```
json
```

```
{
  "name": "complaint-management-frontend",
  "version": "1.0.0",
  "private": true,
  "dependencies": {
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "axios": "^1.4.0"
  },
  "devDependencies": {
    "@babel/core": "^7.22.0",
    "@babel/preset-react": "^7.22.0",
    "babel-loader": "^9.1.0",
    "css-loader": "^6.8.0",
    "html-webpack-plugin": "^5.5.0",
    "style-loader": "^3.3.0",
    "webpack": "^5.88.0",
    "webpack-cli": "^5.1.0",
    "webpack-dev-server": "^4.15.0"
  },
  "scripts": {
    "start": "webpack-dev-server --mode development --open",
    "build": "webpack --mode production"
  }
}
```

2. Webpack Configuration (webpack.config.js)

javascript

```
const HtmlWebpackPlugin = require('html-webpack-plugin');
const path = require('path');

module.exports = {
  entry: './src/index.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'bundle.js'
  },
  module: {
    rules: [
      {
        test: /\.jsx?$/,
        exclude: /node_modules/,
        use: {
          loader: 'babel-loader',
          options: {
            presets: ['@babel/preset-react']
          }
        }
      },
      {
        test: /\.css$/,
        use: ['style-loader', 'css-loader']
      }
    ]
  },
  resolve: {
    extensions: ['.js', '.jsx']
  },
  plugins: [
    new HtmlWebpackPlugin({
      template: './public/index.html'
    })
  ],
  devServer: {
    port: 3000,
    hot: true
  }
};
```

3. HTML Template (public/index.html)

```
html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Complaint Management System</title>
  <style>
    body {
      margin: 0;
      font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Arial, sans-serif;
      background-color: #f5f5f5;
    }
    .container {
      max-width: 1200px;
      margin: 0 auto;
      padding: 20px;
    }
    .header {
      background-color: #2c3e50;
      color: white;
      padding: 1rem;
      text-align: center;
      margin-bottom: 2rem;
    }
  </style>
</head>
<body>
  <div id="root"></div>
</body>
</html>
```

4. API Service (src/services/api.js)

javascript

```
const API_BASE_URL = 'http://localhost:5000/api/v1';

class ApiService {
  async request(endpoint, options = {}) {
    const url = `${API_BASE_URL}${endpoint}`;
    const config = {
      headers: {
        'Content-Type': 'application/json',
      },
      ...options,
    };

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

      if (!response.ok) {
        throw new Error(data.error || 'API request failed');
      }

      return data;
    } catch (error) {
      console.error('API Error:', error);
      throw error;
    }
  }

  // Complaint methods
  async createComplaint(complaintData) {
    return this.request('/complaints', {
      method: 'POST',
      body: JSON.stringify(complaintData),
    });
  }

  async getComplaints(filters = {}) {
    const params = new URLSearchParams(filters);
    return this.request(`/complaints?${params}`);
  }

  async updateComplaintStatus(complaintId, status) {
    return this.request(`/complaints/${complaintId}`, {
      method: 'PUT',
      body: JSON.stringify({ status }),
    });
  }
```

```
  async getCategories() {
    return this.request('/categories');
  }

  async getPriorities() {
    return this.request('/priorities');
  }
}

export default new ApiService();
```

5. Complaint Store (src/store/ComplaintStore.jsx)

javascript

```
import React, { createContext, useContext, useReducer } from 'react';

const ComplaintContext = createContext();

const initialState = {
  complaints: [],
  categories: [],
  priorities: [],
  loading: false,
  error: null,
  filters: {}
};

const complaintReducer = (state, action) => {
  switch (action.type) {
    case 'SET_LOADING':
      return { ...state, loading: action.payload };
    case 'SET_ERROR':
      return { ...state, error: action.payload, loading: false };
    case 'SET_COMPLAINTS':
      return { ...state, complaints: action.payload, loading: false };
    case 'ADD_COMPLAINT':
      return { ...state, complaints: [action.payload, ...state.complaints] };
    case 'UPDATE_COMPLAINT':
      return {
        ...state,
        complaints: state.complaints.map(c =>
          c.id === action.payload.id ? action.payload : c
        )
      };
    case 'SET_CATEGORIES':
      return { ...state, categories: action.payload };
    case 'SET_PRIORITIES':
      return { ...state, priorities: action.payload };
    case 'SET_FILTERS':
      return { ...state, filters: { ...state.filters, ...action.payload } };
    default:
      return state;
  }
};

export const ComplaintProvider = ({ children }) => {
  const [state, dispatch] = useReducer(complaintReducer, initialState);

  return (
    <ComplaintContext.Provider value={{ state, dispatch }}>
```

```
      {children}
    </ComplaintContext.Provider>
  );
};

export const useComplaintStore = () => {
  const context = useContext(ComplaintContext);
  if (!context) {
    throw new Error('useComplaintStore must be used within a ComplaintProvider');
  }
  return context;
};
```

6. Complaint Form Component (src/components/ComplaintForm.jsx)

javascript

```
import React, { useState, useEffect } from 'react';
import { useComplaintStore } from '../store/ComplaintStore';
import ApiService from '../services/api';

const ComplaintForm = ({ onComplaintCreated }) => {
  const { state, dispatch } = useComplaintStore();
  const [formData, setFormData] = useState({
    title: '',
    description: '',
    user_id: 'user123' // In real app, this would come from auth
  });
  const [submitting, setSubmitting] = useState(false);

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

  const loadCategories = async () => {
    try {
      const response = await ApiService.getCategories();
      dispatch({ type: 'SET_CATEGORIES', payload: response.categories });
    } catch (error) {
      dispatch({ type: 'SET_ERROR', payload: error.message });
    }
  };

  const handleInputChange = (e) => {
    const { name, value } = e.target;
    setFormData(prev => ({
      ...prev,
      [name]: value
    }));
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    if (!formData.title.trim() || !formData.description.trim()) {
      alert('Please fill in all required fields');
      return;
    }

    setSubmitting(true);
    try {
      const response = await ApiService.createComplaint(formData);
      dispatch({ type: 'ADD_COMPLAINT', payload: response.complaint });
      setFormData({ title: '', description: '', user_id: 'user123' });
```

```
    if (onComplaintCreated) {
      onComplaintCreated(response.complaint);
    }
  } catch (error) {
    dispatch({ type: 'SET_ERROR', payload: error.message });
  } finally {
    setSubmitting(false);
  }
};

const formStyle = {
  backgroundColor: 'white',
  padding: '2rem',
  borderRadius: '8px',
  boxShadow: '0 2px 4px rgba(0,0,0,0.1)',
  marginBottom: '2rem'
};

const inputStyle = {
  width: '100%',
  padding: '0.75rem',
  border: '1px solid #ddd',
  borderRadius: '4px',
  fontSize: '1rem',
  marginBottom: '1rem'
};

const buttonStyle = {
  backgroundColor: '#3498db',
  color: 'white',
  padding: '0.75rem 1.5rem',
  border: 'none',
  borderRadius: '4px',
  fontSize: '1rem',
  cursor: 'pointer',
  disabled: submitting
};

return (
  <form onSubmit={handleSubmit} style={formStyle}>
    <h2>Submit a Complaint</h2>

    <div>
      <label htmlFor="title">Title *</label>
      <input
        type="text"
        id="title"
```

```
          name="title"
          value={formData.title}
          onChange={handleInputChange}
          style={inputStyle}
          required
        />
      </div>

      <div>
        <label htmlFor="description">Description *</label>
        <textarea
          id="description"
          name="description"
          value={formData.description}
          onChange={handleInputChange}
          rows="4"
          style={{ ...inputStyle, resize: 'vertical' }}
          required
        />
      </div>

      <button
        type="submit"
        style={buttonStyle}
        disabled={submitting}
      >
        {submitting ? 'Submitting...' : 'Submit Complaint'}
      </button>
    </form>
  );
};

export default ComplaintForm;
```

7. Complaint Card Component (src/components/ComplaintCard.jsx)

javascript

```
import React from 'react';
import ApiService from '../services/api';
import { useComplaintStore } from '../store/ComplaintStore';

const ComplaintCard = ({ complaint }) => {
  const { dispatch } = useComplaintStore();

  const handleStatusChange = async (newStatus) => {
    try {
      const response = await ApiService.updateComplaintStatus(complaint.id, newStatus);
      dispatch({ type: 'UPDATE_COMPLAINT', payload: response.complaint });
    } catch (error) {
      dispatch({ type: 'SET_ERROR', payload: error.message });
    }
  };

  const getPriorityColor = (priority) => {
    const colors = {
      high: '#dc3545',
      medium: '#ffc107',
      low: '#28a745'
    };
    return colors[priority] || '#6c757d';
  };

  const getStatusColor = (status) => {
    const colors = {
      open: '#007bff',
      in_progress: '#ffc107',
      resolved: '#28a745',
      closed: '#6c757d'
    };
    return colors[status] || '#6c757d';
  };

  const cardStyle = {
    backgroundColor: 'white',
    padding: '1.5rem',
    borderRadius: '8px',
    boxShadow: '0 2px 4px rgba(0,0,0,0.1)',
    marginBottom: '1rem',
    border: `3px solid ${getPriorityColor(complaint.priority_id)}`
  };

  const headerStyle = {
    display: 'flex',
```

```
  justifyContent: 'space-between',
  alignItems: 'center',
  marginBottom: '1rem'
};

const badgeStyle = {
  padding: '0.25rem 0.75rem',
  borderRadius: '12px',
  fontSize: '0.875rem',
  fontWeight: 'bold',
  color: 'white'
};

const selectStyle = {
  padding: '0.5rem',
  borderRadius: '4px',
  border: '1px solid #ddd'
};

return (
  <div style={cardStyle}>
    <div style={headerStyle}>
      <h3 style={{ margin: 0 }}>{complaint.title}</h3>
      <div style={{ display: 'flex', gap: '0.5rem' }}>
        <span
          style={{
            ...badgeStyle,
            backgroundColor: getPriorityColor(complaint.priority_id)
          }}
        >
          {complaint.priority_id?.toUpperCase()}
        </span>
        <span
          style={{
            ...badgeStyle,
            backgroundColor: getStatusColor(complaint.status)
          }}
        >
          {complaint.status?.toUpperCase()}
        </span>
      </div>
    </div>

    <p style={{ marginBottom: '1rem', color: '#666' }}>
      {complaint.description}
    </p>
```

```
      <div style={{ display: 'flex', justifyContent: 'space-between', alignItems: 'center' }}>
        <div style={{ fontSize: '0.875rem', color: '#666' }}>
          <strong>Category:</strong> {complaint.category_id || 'General'}<br />
          <strong>Created:</strong> {new Date(complaint.created_at).toLocaleDateString()}
        </div>

        <select
          value={complaint.status}
          onChange={(e) => handleStatusChange(e.target.value)}
          style={selectStyle}
        >
          <option value="open">Open</option>
          <option value="in_progress">In Progress</option>
          <option value="resolved">Resolved</option>
          <option value="closed">Closed</option>
        </select>
      </div>
    </div>
  );
};

export default ComplaintCard;
```

8. Complaint List Component (src/components/ComplaintList.jsx)

javascript

```
import React, { useEffect, useState } from 'react';
import { useComplaintStore } from '../store/ComplaintStore';
import ComplaintCard from './ComplaintCard';
import ApiService from '../services/api';

const ComplaintList = () => {
  const { state, dispatch } = useComplaintStore();
  const [filters, setFilters] = useState({
    status: '',
    category: '',
    priority: ''
  });

  useEffect(() => {
    loadComplaints();
    loadCategories();
    loadPriorities();
  }, []);

  useEffect(() => {
    loadComplaints();
  }, [filters]);

  const loadComplaints = async () => {
    dispatch({ type: 'SET_LOADING', payload: true });
    try {
      const activeFilters = Object.fromEntries(
        Object.entries(filters).filter(([_, value]) => value)
      );
      const response = await ApiService.getComplaints(activeFilters);
      dispatch({ type: 'SET_COMPLAINTS', payload: response.complaints });
    } catch (error) {
      dispatch({ type: 'SET_ERROR', payload: error.message });
    }
  };

  const loadCategories = async () => {
    try {
      const response = await ApiService.getCategories();
      dispatch({ type: 'SET_CATEGORIES', payload: response.categories });
    } catch (error) {
      console.error('Failed to load categories:', error);
    }
  };

  const loadPriorities = async () => {
```

```
  try {
    const response = await ApiService.getPriorities();
    dispatch({ type: 'SET_PRIORITIES', payload: response.priorities });
  } catch (error) {
    console.error('Failed to load priorities:', error);
  }
};

const handleFilterChange = (filterType, value) => {
  setFilters(prev => ({
    ...prev,
    [filterType]: value
  }));
};

const filterStyle = {
  backgroundColor: 'white',
  padding: '1rem',
  borderRadius: '8px',
  boxShadow: '0 2px 4px rgba(0,0,0,0.1)',
  marginBottom: '1rem',
  display: 'flex',
  gap: '1rem',
  flexWrap: 'wrap'
};

const selectStyle = {
  padding: '0.5rem',
  borderRadius: '4px',
  border: '1px solid #ddd',
  minWidth: '120px'
};

if (state.loading) {
  return <div style={{ textAlign: 'center', padding: '2rem' }}>Loading...</div>;
}

if (state.error) {
  return (
    <div style={{
      color: '#dc3545',
      textAlign: 'center',
      padding: '2rem',
      backgroundColor: 'white',
      borderRadius: '8px'
    }}>
      Error: {state.error}
```

```
    </div>
  );
}

return (
  <div>
    <h2>All Complaints ({state.complaints.length})</h2>

    {/* Filters */}
    <div style={filterStyle}>
      <div>
        <label htmlFor="status-filter">Status:</label>
        <select
          id="status-filter"
          value={filters.status}
          onChange={(e) => handleFilterChange('status', e.target.value)}
          style={selectStyle}
        >
          <option value="">All Status</option>
          <option value="open">Open</option>
          <option value="in_progress">In Progress</option>
          <option value="resolved">Resolved</option>
          <option value="closed">Closed</option>
        </select>
      </div>

      <div>
        <label htmlFor="category-filter">Category:</label>
        <select
          id="category-filter"
          value={filters.category}
          onChange={(e) => handleFilterChange('category', e.target.value)}
          style={selectStyle}
        >
          <option value="">All Categories</option>
          {state.categories.map(category => (
            <option key={category.id} value={category.id}>
              {category.name}
            </option>
          ))}
        </select>
      </div>

      <div>
        <label htmlFor="priority-filter">Priority:</label>
        <select
          id="priority-filter"
```

```
          value={filters.priority}
          onChange={(e) => handleFilterChange('priority', e.target.value)}
          style={selectStyle}
        >
          <option value="">All Priorities</option>
          {state.priorities.map(priority => (
            <option key={priority.id} value={priority.id}>
              {priority.name}
            </option>
          ))}
        </select>
      </div>

      {(filters.status || filters.category || filters.priority) && (
        <button
          onClick={() => setFilters({ status: '', category: '', priority: '' })}
          style={{
            padding: '0.5rem 1rem',
            backgroundColor: '#6c757d',
            color: 'white',
            border: 'none',
            borderRadius: '4px',
            cursor: 'pointer'
          }}
        >
          Clear Filters
        </button>
      )}
    </div>

    {/* Complaints List */}
    {state.complaints.length === 0 ? (
      <div style={{
        textAlign: 'center',
        padding: '3rem',
        backgroundColor: 'white',
        borderRadius: '8px'
      }}>
        <h3>No complaints found</h3>
        <p>Try adjusting your filters or submit a new complaint.</p>
      </div>
    ) : (
      <div>
        {state.complaints.map(complaint => (
          <ComplaintCard key={complaint.id} complaint={complaint} />
        ))}
      </div>
```

```
      )}
    </div>
  );
};

export default ComplaintList;
```

9. Main App Component (src/App.jsx)

javascript

```jsx
import React, { useState } from 'react';
import { ComplaintProvider } from './store/ComplaintStore';
import ComplaintForm from './components/ComplaintForm';
import ComplaintList from './components/ComplaintList';

const App = () => {
  const [activeTab, setActiveTab] = useState('list');

  const tabStyle = {
    padding: '0.75rem 1.5rem',
    backgroundColor: 'transparent',
    border: 'none',
    cursor: 'pointer',
    fontSize: '1rem',
    borderBottom: '2px solid transparent'
  };

  const activeTabStyle = {
    ...tabStyle,
    borderBottom: '2px solid #3498db',
    color: '#3498db',
    fontWeight: 'bold'
  };

  const navStyle = {
    backgroundColor: 'white',
    marginBottom: '2rem',
    boxShadow: '0 2px 4px rgba(0,0,0,0.1)'
  };

  return (
    <ComplaintProvider>
      <div>
        <div className="header">
          <h1>Complaint Management System</h1>
          <p>Submit and track complaints efficiently</p>
        </div>

        <div className="container">
          <nav style={navStyle}>
            <button
              style={activeTab === 'list' ? activeTabStyle : tabStyle}
              onClick={() => setActiveTab('list')}
            >
              All Complaints
            </button>
```

```
          <button
            style={activeTab === 'create' ? activeTabStyle : tabStyle}
            onClick={() => setActiveTab('create')}
          >
            Submit Complaint
          </button>
        </nav>

        {activeTab === 'create' && (
          <ComplaintForm onComplaintCreated={() => setActiveTab('list')} />
        )}

        {activeTab === 'list' && <ComplaintList />}
      </div>
    </div>
  </ComplaintProvider>
 );
};

export default App;
```

10. Entry Point (src/index.js)

javascript

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(<App />);
```

Docker Deployment

Docker Compose (docker-compose.yml)

yaml

```yaml
version: '3.8'

services:
  backend:
    build: ./backend
    ports:
      - "5000:5000"
    environment:
      - FLASK_ENV=development
      - DATABASE_URL=sqlite:///complaints.db
      - ENABLE_AI_CLASSIFICATION=true
    volumes:
      - ./backend/config:/app/config
      - ./backend/data:/app/data

  frontend:
    build: ./frontend
    ports:
      - "3000:3000"
    depends_on:
      - backend
    environment:
      - REACT_APP_API_URL=http://localhost:5000/api/v1

networks:
  default:
    driver: bridge
```

Backend Dockerfile (backend/Dockerfile)

dockerfile

```dockerfile
FROM python:3.11-slim

WORKDIR /app

COPY requirements.txt .
RUN pip install -r requirements.txt

COPY . .

EXPOSE 5000

CMD ["python", "run.py"]
```

Frontend Dockerfile (frontend/Dockerfile)

```dockerfile
FROM node:18-slim

WORKDIR /app

COPY package*.json ./
RUN npm install

COPY . .

EXPOSE 3000

CMD ["npm", "start"]
```

Execution Instructions

Option 1: Manual Setup

Backend Setup:

```bash
# Navigate to backend directory
cd backend

# Create virtual environment
python -m venv venv

# Activate virtual environment
# On Windows:
venv\Scripts\activate
# On Mac/Linux:
source venv/bin/activate

# Install dependencies
pip install -r requirements.txt

# Run the backend server
python run.py
```

Frontend Setup:

```bash
```

```
# Navigate to frontend directory (in new terminal)
cd frontend

# Install dependencies
npm install

# Start development server
npm start
```

Option 2: Docker Setup

```
bash

# From project root directory
docker-compose up --build
```

Access the Application:

- Frontend: http://localhost:3000
- Backend API: http://localhost:5000/api/v1

Key Features Implemented:

1. **Configuration-Driven Architecture**
   - YAML-based categories, priorities, and features
   - Environment-specific configurations
2. **AI-Powered Classification**
   - Simple keyword-based classifier
   - Automatic category and priority assignment
3. **Flexible Database Schema**
   - JSON metadata fields for extensibility
   - Migration-ready structure
4. **RESTful API**
   - Versioned endpoints
   - Comprehensive CRUD operations
5. **React Frontend**
   - Component-based architecture
   - Context-based state management
   - Real-time updates

6. **Admin Features**
    - Status management
    - Filtering and search
    - Category management

Testing the System:

1. Submit complaints with different descriptions
2. Watch AI classification in action
3. Filter complaints by status, category, priority
4. Update complaint statuses
5. View real-time updates

This implementation provides a solid foundation for a maintainable complaint management system that can be easily extended and configured.