CivicDoc — AI-Powered Public Issue Prioritization & Governance Automation System

This repository scaffold contains a working minimal prototype for CivicDoc (FastAPI backend, simple HTML/JS frontend, ML classifier using TF-IDF + XGBoost, OCR/PDF handling, MongoDB storage, Jinja2 templates). The goal is a clear, extensible structure you can run and expand.

File structure

```
CivicDoc/
├── README.md
├── requirements.txt
├── Dockerfile
├── docker-compose.yml
├── deploy.sh
├── data/
│   ├── raw/                # raw complaint text, PDFs etc (not in git)
│   └── models/             # trained models saved here
├── app/
│   ├── main.py             # FastAPI app entry
│   ├── config.py           # configuration & settings
│   ├── db.py               # MongoDB connection
│   ├── models.py           # Pydantic models & DB schemas
│   ├── routes/
│   │   ├── complaints.py   # complaint endpoints
│   │   ├── documents.py    # document generation endpoints
│   │   └── health.py
│   ├── ml/
│   │   ├── train.py        # training pipeline (TF-IDF + XGBoost)
│   │   └── predict.py      # prediction wrapper
│   ├── ocr.py              # Tesseract wrapper for images & PDFs
│   ├── pdf_parser.py       # PyMuPDF helpers for PDFs & circular parsing
│   ├── templates/
│   │   ├── rti_template.html
│   │   └── workorder_template.html
│   └── utils.py            # helpers (priority scoring, routing, multilingual)
├── frontend/
│   ├── index.html
│   ├── styles.css
│   └── js/
│       └── app.js
```

```
└── sample_requests/
    └── example_complaint.json
```

How to use (quick)

1. Create a Python venv: `python -m venv .venv && source .venv/bin/activate`
2. Install: `pip install -r requirements.txt`
3. Start MongoDB (docker / local), set `MONGODB_URI` in env or `app/config.py`.
4. (Optional) Train model: `python app/ml/train.py --data data/raw/complaints.csv`
5. Start API: `uvicorn app.main:app --reload`
6. Open `frontend/index.html` in browser (or serve via simple HTTP server).

Key files (code)

requirements.txt

```
fastapi
uvicorn[standard]
motor
pydantic
python-multipart
pymupdf
pytesseract
pillow
spacy
nltk
scikit-learn
xgboost
jinja2
python-dotenv
pyyaml
```

---

app/config.py

```python
from pydantic import BaseSettings

class Settings(BaseSettings):
    app_name: str = "CivicDoc"
    mongodb_uri: str = "mongodb://localhost:27017"
    mongodb_db: str = "civicdoc"
    model_path: str = "data/models/classifier.pkl"
    tfidf_path: str = "data/models/tfidf.pkl"
    supported_langs: list = ["en","hi","kn","mr","ta","te"]

    class Config:
        env_file = ".env"

settings = Settings()
```

---

app/db.py

```
import motor.motor_asyncio
from .config import settings

client = motor.motor_asyncio.AsyncIOMotorClient(settings.mongodb_uri)
db = client[settings.mongodb_db]

# convenience collections
complaints_col = db.get_collection("complaints")
```

---

app/models.py

```
from pydantic import BaseModel, Field
from typing import Optional, List
from datetime import datetime

class ComplaintCreate(BaseModel):
    citizen_name: Optional[str]
    contact: Optional[str]
    message: str
    source: Optional[str] = "web"
    location: Optional[str]

class ComplaintDB(ComplaintCreate):
    id: Optional[str]
    category: str
    priority_score: int
    urgency: str
    routed_to: str
    status: str = "pending"
    created_at: datetime = Field(default_factory=datetime.utcnow)
```

```
class DocumentRequest(BaseModel):
    doc_type: str  # rti, application, notice, workorder
    complaint_id: Optional[str]
    circular_text: Optional[str]
    language: str = "en"
```

---

app/utils.py

```
from typing import Tuple

CATEGORIES =
["sewage","garbage","water","roads","electricity","pollution","safety"]
DEPARTMENTS = {
    "sewage":"Sanitation Dept",
    "garbage":"Sanitation Dept",
    "water":"Water Board",
    "roads":"PWD",
    "electricity":"Electricity Dept",
    "pollution":"Environment Dept",
    "safety":"Police/Traffic"
}


def priority_score_from_probs(category_probs: dict, urgency_flag: bool,
severity_est: int) -> int:
    """Compute 0-100 priority score. severity_est in 0-10."""
    base = max(category_probs.values()) * 60  # category weight
    urg = 30 if urgency_flag else 0
    sev = (severity_est / 10) * 10
    score = int(min(100, base + urg + sev))
    return score


def route_to_department(category: str) -> str:
    return DEPARTMENTS.get(category, "General Municipal Office")
```

---

app/ml/train.py

```
"""
Train a simple TF-IDF + XGBoost classifier for complaint category.
This script expects a CSV with columns: text, category, urgency (0/1), severity
(0-10)
```

```python
"""
import argparse
import joblib
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
from pathlib import Path

from app.config import settings


def main(data_csv):
    df = pd.read_csv(data_csv)
    X = df['text'].fillna("")
    y = df['category']

    tfidf = TfidfVectorizer(max_features=10000, ngram_range=(1,2))
    Xv = tfidf.fit_transform(X)

    X_train, X_val, y_train, y_val = train_test_split(Xv, y, test_size=0.2,
random_state=42)
    clf = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')
    clf.fit(X_train, y_train)

    preds = clf.predict(X_val)
    print(classification_report(y_val, preds))

    Path(settings.model_path).parent.mkdir(parents=True, exist_ok=True)
    joblib.dump(clf, settings.model_path)
    joblib.dump(tfidf, settings.tfidf_path)
    print("Saved model and tfidf")


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--data', required=True)
    args = parser.parse_args()
    main(args.data)
```

app/ml/predict.py

```python
import joblib
from typing import Dict
from .config import settings

clf = None
tfidf = None


def load():
    global clf, tfidf
    if clf is None:
        clf = joblib.load(settings.model_path)
    if tfidf is None:
        tfidf = joblib.load(settings.tfidf_path)


def predict(text: str) -> Dict:
    load()
    X = tfidf.transform([text])
    pred = clf.predict(X)[0]
    probs = clf.predict_proba(X)[0]
    labels = clf.classes_.tolist()
    prob_map = {labels[i]: float(probs[i]) for i in range(len(labels))}
    return {"category": pred, "probs": prob_map}
```

---

app/ocr.py

```python
import pytesseract
from PIL import Image
import fitz  # PyMuPDF
from io import BytesIO


def image_to_text(image_bytes: bytes) -> str:
    img = Image.open(BytesIO(image_bytes))
    return pytesseract.image_to_string(img, lang='eng')


def pdf_to_text(pdf_path: str) -> str:
    text = []
    doc = fitz.open(pdf_path)
    for page in doc:
        text.append(page.get_text())
    return "\n".join(text)
```

app/pdf_parser.py

```
# simple heuristics to extract eligibility, deadlines, contact info from
circular text
import re

def extract_deadlines(text: str) -> list:
    # naive: find date-like patterns
    patterns = re.findall(r"\b\d{1,2}[\-/ ](?:Jan|Feb|Mar|Apr|May|Jun|Jul|Aug|
Sep|Oct|Nov|Dec|\d{2,4})[\-/ ]?\d{2,4}\b", text, flags=re.I)
    return patterns


def extract_eligibility(text: str) -> str:
    # look for 'eligible' lines
    m = re.search(r"(eligible[^.\n]+[.\n])", text, flags=re.I)
    return m.group(1) if m else ""
```

app/routes/complaints.py

```
from fastapi import APIRouter, HTTPException
from app.models import ComplaintCreate, ComplaintDB
from app.db import complaints_col
from app.ml.predict import predict
from app.utils import priority_score_from_probs, route_to_department

router = APIRouter()

@router.post('/complaints', response_model=ComplaintDB)
async def create_complaint(payload: ComplaintCreate):
    # classify
    res = predict(payload.message)
    category = res['category']
    probs = res['probs']
    urgency_flag = 'urgent' in payload.message.lower() or
max(probs.values())<0.4
    severity_est = 5
    score = priority_score_from_probs(probs, urgency_flag, severity_est)
    routed = route_to_department(category)
    doc = payload.dict()
    doc.update({
        'category': category,
        'priority_score': score,
        'urgency': 'high' if urgency_flag else 'normal',
        'routed_to': routed,
        'status': 'pending'
    })
    res = await complaints_col.insert_one(doc)
    doc['id'] = str(res.inserted_id)
    return doc

@router.get('/complaints/{cid}')
async def get_complaint(cid: str):
    doc = await complaints_col.find_one({"_id": cid})
    if not doc:
        raise HTTPException(status_code=404, detail='Not found')
    doc['id'] = str(doc['_id'])
    return doc
```

---

app/routes/documents.py

```
from fastapi import APIRouter
from fastapi.responses import HTMLResponse
from jinja2 import Environment, FileSystemLoader
from app.models import DocumentRequest
from app.db import complaints_col

router = APIRouter()
env = Environment(loader=FileSystemLoader('app/templates'))

@router.post('/generate', response_class=HTMLResponse)
async def generate(req: DocumentRequest):
    # simple document generation
    template_name = 'rti_template.html' if req.doc_type=='rti' else
'workorder_template.html'
    tmpl = env.get_template(template_name)
    context = {"req": req.dict()}
    if req.complaint_id:
        c = await complaints_col.find_one({"_id": req.complaint_id})
        context['complaint'] = c
    html = tmpl.render(**context)
    return HTMLResponse(content=html)
```

app/main.py

```
from fastapi import FastAPI
from app.routes import complaints, documents
from fastapi.middleware.cors import CORSMiddleware

app = FastAPI(title='CivicDoc')
app.add_middleware(CORSMiddleware, allow_origins=['*'], allow_methods=['*'],
allow_headers=['*'])

app.include_router(complaints.router, prefix='/api')
app.include_router(documents.router, prefix='/api')

@app.get('/')
def health():
    return {"ok": True}
```

app/templates/rti_template.html

```
<!doctype html>
<html>
  <body>
    <h2>Right to Information (RTI) Application</h2>
    <p>Date: {{ req.get('date') or '' }}</p>
    <p>To: Public Information Officer</p>
    <p>Subject: Request for information regarding complaint ID
{{ req.get('complaint_id') }}</p>
    <p>Details: Please provide the following information...</p>
  </body>
</html>
```

frontend/index.html

```
<!doctype html>
<html>
<head>
  <meta charset="utf-8" />
  <title>CivicDoc — Demo</title>
  <link rel="stylesheet" href="styles.css">
</head>
```

```
<body>
  <h1>CivicDoc — Submit Complaint</h1>
  <textarea id="message" rows="6" cols="60" placeholder="Describe the
issue..."></textarea><br>
  <input id="name" placeholder="Name" />
  <input id="contact" placeholder="Contact" />
  <button id="send">Submit</button>

  <pre id="resp"></pre>
  <script src="js/app.js"></script>
</body>
</html>
```

---

frontend/js/app.js

```
const apiBase = 'http://localhost:8000/api'

document.getElementById('send').addEventListener('click', async ()=>{
  const message = document.getElementById('message').value
  const name = document.getElementById('name').value
  const contact = document.getElementById('contact').value
  const payload = { message, citizen_name: name, contact }
  const res = await fetch(apiBase + '/complaints', {
    method: 'POST', headers: {'Content-Type':'application/json'}, body:
JSON.stringify(payload)
  })
  const data = await res.json()
  document.getElementById('resp').innerText = JSON.stringify(data, null, 2)
})
```

Dockerfile

```
FROM python:3.11-slim
WORKDIR /app
COPY . /app
RUN pip install -r requirements.txt
EXPOSE 8000
CMD ["uvicorn","app.main:app","--host","0.0.0.0","--port","8000"]
```

README.md (short)

```
# CivicDoc

Prototype for AI-driven civic complaint automation.

## Run

1. Install requirements
2. Start MongoDB
3. Train model (optional)
4. Run: `uvicorn app.main:app --reload`

Open frontend/index.html for demo UI.
```

Notes & Next steps

- **Multilingual**: integrate `langdetect` + Marian or Transformers for translations; for a lightweight setup, use `googletrans` or `indic-transliteration` (be mindful of licensing).
- **Security**: add auth, rate-limiting, input sanitization, and validation before deployment.
- **Production**: add async batching for OCR/PDF jobs (Celery/RQ) and monitoring.
- **Improvements**: fine-grained priority scoring using entity extraction (Spacy), geo-parsing for location, and feedback loop to re-train the classifier.

If you want, I can: - generate a `docker-compose.yml` for Mongo + app; - add unit tests for the ML pipeline; - expand templates for multilingual RTI generation (Hindi/Marathi/Kannada/Tamil/Telugu) using translation stubs.

Tell me which part you want next and I will expand it.