# **Project Structure**

fuzzy\_traffic\_control/

│

├── backend/

│ ├── app.py # Main backend app (Flask/FastAPI)

│ ├── fuzzy\_logic.py # Core fuzzy logic implementation

│ ├── models.py # Database ORM models

│ ├── api.py # API endpoints

│ ├── scheduler.py # Real-time data polling/scheduling

│ └── utils.py # Helper functions

│

├── frontend/

│ ├── static/

│ ├── templates/

│ └── main.js # UI logic (e.g., for dashboard)

│

├── tests/

│ ├── test\_fuzzy\_logic.py

│ └── test\_api.py

│

├── deployment/

│ └── docker-compose.yml # For containerized deployment

│

├── requirements.txt

├── README.md

└── docs/

## **Key Module 1: Fuzzy Logic Engine**

****Explanation:****  
Implements the core fuzzy inference that determines traffic signal duration based on inputs (pedestrian density, vehicle flow, weather).

****Sample Code (Python, using**scikit-fuzzy**):****

import numpy as npimport skfuzzy as fuzzfrom skfuzzy import control as ctrl

# Define fuzzy variablespedestrian\_density = ctrl.Antecedent(np.arange(0, 101, 1), 'pedestrian\_density')vehicle\_flow = ctrl.Antecedent(np.arange(0, 101, 1), 'vehicle\_flow')weather = ctrl.Antecedent(np.arange(0, 4, 1), 'weather') # 0: Clear, 1: Rain, 2: Fog, 3: Snowgreen\_time = ctrl.Consequent(np.arange(10, 181, 1), 'green\_time')

# Membership functionspedestrian\_density['low'] = fuzz.trimf(pedestrian\_density.universe, [0, 0, 50])pedestrian\_density['medium'] = fuzz.trimf(pedestrian\_density.universe, [20, 50, 80])pedestrian\_density['high'] = fuzz.trimf(pedestrian\_density.universe, [60, 100, 100])

vehicle\_flow['low'] = fuzz.trimf(vehicle\_flow.universe, [0, 0, 40])vehicle\_flow['medium'] = fuzz.trimf(vehicle\_flow.universe, [20, 50, 80])vehicle\_flow['high'] = fuzz.trimf(vehicle\_flow.universe, [60, 100, 100])

weather['good'] = fuzz.trimf(weather.universe, [0, 0, 1])weather['moderate'] = fuzz.trimf(weather.universe, [1, 2, 3])weather['bad'] = fuzz.trimf(weather.universe, [2, 3, 3])

green\_time['short'] = fuzz.trimf(green\_time.universe, [10, 10, 60])green\_time['medium'] = fuzz.trimf(green\_time.universe, [40, 90, 140])green\_time['long'] = fuzz.trimf(green\_time.universe, [120, 180, 180])

# Rulesrule1 = ctrl.Rule(pedestrian\_density['high'] & vehicle\_flow['low'] & weather['good'], green\_time['long'])rule2 = ctrl.Rule(pedestrian\_density['low'] & vehicle\_flow['high'] & weather['good'], green\_time['long'])rule3 = ctrl.Rule(weather['bad'], green\_time['short'])rule4 = ctrl.Rule(pedestrian\_density['medium'] | vehicle\_flow['medium'], green\_time['medium'])

# Control Systemsignal\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3, rule4])signal\_sim = ctrl.ControlSystemSimulation(signal\_ctrl)

def compute\_green\_time(pedestrian, vehicle, weather\_code):

signal\_sim.input['pedestrian\_density'] = pedestrian

signal\_sim.input['vehicle\_flow'] = vehicle

signal\_sim.input['weather'] = weather\_code

signal\_sim.compute()

return signal\_sim.output['green\_time']

## **Key Module 2: Real-Time Data Acquisition**

****Explanation:****  
Collects sensor data (pedestrian counts, vehicle flow, weather) from APIs or hardware interfaces at regular intervals.

****Sample Code:****

import requestsfrom datetime import datetime

def fetch\_sensor\_data(intersection\_id):

# Example: Fetch from hardware API or simulated endpoint

resp = requests.get(f'http://localhost:5001/api/sensors/{intersection\_id}')

return resp.json()

def fetch\_weather\_data(intersection\_id):

# Example: Use OpenWeatherMap API (replace with real API key/logic)

# resp = requests.get(f'https://api.weather.com/v3/wx/conditions/current?...')

# return resp.json()

# Simulated:

return {'temperature': 27, 'rainfall': 0, 'weather\_type': 'Clear', 'weather\_code': 0}

def aggregate\_inputs(intersection\_id):

sensor = fetch\_sensor\_data(intersection\_id)

weather = fetch\_weather\_data(intersection\_id)

return {

'pedestrian\_density': sensor['pedestrians'],

'vehicle\_flow': sensor['vehicles'],

'weather\_code': weather['weather\_code']

}

## **Key Module 3: API Layer**

****Explanation:****  
Exposes RESTful endpoints for the frontend/UI and data ingestion.

****Sample Code (Flask):****

from flask import Flask, request, jsonifyfrom fuzzy\_logic import compute\_green\_timefrom scheduler import aggregate\_inputs

app = Flask(\_\_name\_\_)

@app.route('/api/traffic\_signal', methods=['POST'])def traffic\_signal():

data = request.json

# Or aggregate from real sensors:

# inputs = aggregate\_inputs(data['intersection\_id'])

pedestrian = data['pedestrian\_density']

vehicle = data['vehicle\_flow']

weather\_code = data['weather\_code']

green\_time = compute\_green\_time(pedestrian, vehicle, weather\_code)

return jsonify({'green\_time': green\_time})

# Health check@app.route('/api/health', methods=['GET'])def health():

return jsonify({'status': 'ok'})

## **Database Design (ER Diagram + Code/Queries)**

****ER Diagram (description):****

* ****Intersection**** (1) --- (M) ****Sensor****
* ****Sensor**** (1) --- (M) ****PedestrianCount****
* ****Sensor**** (1) --- (M) ****VehicleCount****
* ****Intersection**** (1) --- (M) ****Weather****
* ****Intersection**** (1) --- (M) ****SignalState****

****SQL Example:****

CREATE TABLE Intersection (

id INT PRIMARY KEY AUTO\_INCREMENT,

name VARCHAR(100),

latitude FLOAT,

longitude FLOAT

);

CREATE TABLE Sensor (

id INT PRIMARY KEY AUTO\_INCREMENT,

intersection\_id INT,

type ENUM('pedestrian', 'vehicle'),

location VARCHAR(100),

FOREIGN KEY (intersection\_id) REFERENCES Intersection(id)

);

CREATE TABLE PedestrianCount (

id INT PRIMARY KEY AUTO\_INCREMENT,

sensor\_id INT,

timestamp DATETIME,

count INT,

FOREIGN KEY (sensor\_id) REFERENCES Sensor(id)

);

CREATE TABLE VehicleCount (

id INT PRIMARY KEY AUTO\_INCREMENT,

sensor\_id INT,

timestamp DATETIME,

count INT,

FOREIGN KEY (sensor\_id) REFERENCES Sensor(id)

);

CREATE TABLE Weather (

id INT PRIMARY KEY AUTO\_INCREMENT,

intersection\_id INT,

timestamp DATETIME,

temperature FLOAT,

rainfall FLOAT,

weather\_type VARCHAR(50),

weather\_code INT,

FOREIGN KEY (intersection\_id) REFERENCES Intersection(id)

);

CREATE TABLE SignalState (

id INT PRIMARY KEY AUTO\_INCREMENT,

intersection\_id INT,

timestamp DATETIME,

green\_time INT,

yellow\_time INT,

red\_time INT,

fuzzy\_output VARCHAR(100),

FOREIGN KEY (intersection\_id) REFERENCES Intersection(id)

);

## **API Design (Endpoints, Code Snippets)**

Endpoints:

* POST /api/traffic\_signal — Input: densities, weather, Output: computed green time
* GET /api/health — Health check
* GET /api/intersections/<id>/status — Current signal state
* POST /api/sensor\_data — Ingest sensor readings

OpenAPI Example:

paths:

/api/traffic\_signal:

post:

summary: Compute optimal green time

requestBody:

content:

application/json:

schema:

type: object

properties:

pedestrian\_density: { type: integer }

vehicle\_flow: { type: integer }

weather\_code: { type: integer }

responses:

'200':

description: Green time calculated

content:

application/json:

schema:

type: object

properties:

green\_time: { type: number }

## **Frontend Implementation (UI, Sample Code)**

****Explanation:****  
Simple dashboard UI displaying intersection status, live signal times, and trends.

****Sample Code (HTML + JS):****

<!DOCTYPE html><html><head>

<title>Traffic Control Dashboard</title></head><body>

<h1>Intersection Dashboard</h1>

<div>

<label>Pedestrian Density: <input id="ped" type="number" /></label>

<label>Vehicle Flow: <input id="veh" type="number" /></label>

<label>Weather:

<select id="weather">

<option value="0">Clear</option>

<option value="1">Rain</option>

<option value="2">Fog</option>

<option value="3">Snow</option>

</select>

</label>

<button onclick="compute()">Compute Green Time</button>

</div>

<pre id="output"></pre>

<script>

async function compute() {

const res = await fetch('/api/traffic\_signal', {

method: 'POST',

headers: {'Content-Type': 'application/json'},

body: JSON.stringify({

pedestrian\_density: parseInt(document.getElementById('ped').value),

vehicle\_flow: parseInt(document.getElementById('veh').value),

weather\_code: parseInt(document.getElementById('weather').value)

})

});

document.getElementById('output').innerText = JSON.stringify(await res.json(), null, 2);

}

</script></body></html>

## **Backend Implementation (Logic, Sample Code)**

****See Key Modules above.****  
The backend (Flask/FastAPI) integrates the fuzzy logic, exposes REST endpoints, and interacts with the database.

## **Data Processing (with code)**

****Explanation:****  
Transforms raw sensor input for the fuzzy engine, stores time-series data.

****Sample Code:****

import datetime

def preprocess\_sensor\_data(raw):

# Ensure all fields present and non-negative

return {

'pedestrian\_density': max(0, raw.get('pedestrians', 0)),

'vehicle\_flow': max(0, raw.get('vehicles', 0)),

'timestamp': raw.get('timestamp', datetime.datetime.now())

}

## **Testing Approach (Unit/Integration, with code)**

from backend.fuzzy\_logic import compute\_green\_time

def test\_green\_time\_high\_pedestrians():

result = compute\_green\_time(90, 30, 0)

assert result > 100 # Should be long green time

def test\_green\_time\_bad\_weather():

result = compute\_green\_time(50, 50, 3)

assert result < 60 # Should be short green time

## **Error Handling (Examples)**

@app.errorhandler(Exception)def handle\_error(e):

return jsonify({'error': str(e)}), 500

* Validate all user inputs (e.g., negative densities)
* Log all exceptions for audit/troubleshooting

## **Security Features (with code)**

* Use HTTPS for data in transit
* Input validation to prevent injection
* Authentication/authorization for admin endpoints

from flask import abort

def require\_api\_key(request):

if request.headers.get('x-api-key') != "SECRET":

abort(401)

## **Sample Input/Output**

****Input JSON:****

{

"pedestrian\_density": 70,

"vehicle\_flow": 30,

"weather\_code": 1

}

****Output JSON:****

{

"green\_time": 120.5

}

## **Performance Optimization (if any)**

* Use asynchronous endpoints (FastAPI/async Flask)
* Cache fuzzy computation results for repeated inputs
* Index timestamp fields in DB for fast queries

## **Deployment Script/Process**

****Docker Compose Example:****

version: '3'services:

backend:

build: ../backend

ports:

- "5000:5000"

env\_file:

- ../backend/.env

db:

image: mysql:8

environment:

MYSQL\_ROOT\_PASSWORD: password

MYSQL\_DATABASE: traffic

ports:

- "3306:3306"

## **Version Control Practices**

* Use Git; commit each module/feature separately
* Use feature branches, PRs, and code reviews
* Example .gitignore:

\_\_pycache\_\_/

\*.pyc

.env

.db

## **Documentation/Comments Example**

"""This module implements the fuzzy logic engine for traffic signal calculation.Inputs: pedestrian\_density, vehicle\_flow, weather\_codeOutput: optimal green\_time (seconds)"""def compute\_green\_time(...):

"""Calculate green time based on fuzzy inference."""

...

## **Key Challenges in Code**

* Real-time data consistency and error handling
* Accurate fuzzy membership design and rules
* Handling missing/incomplete sensor data
* Scaling for multiple intersections

## **Summary Table/List of Major Functions/Classes**

| **Module** | **Function/Class** | **Purpose** |
| --- | --- | --- |
| fuzzy\_logic.py | compute\_green\_time | Fuzzy inference for green time |
| scheduler.py | aggregate\_inputs | Fetch & aggregate sensor+weather data |
| api.py | /api/traffic\_signal | Compute green time via REST API |
| models.py | DB models | ORM for all tables |
| utils.py | preprocess\_sensor\_data | Clean/validate input data |