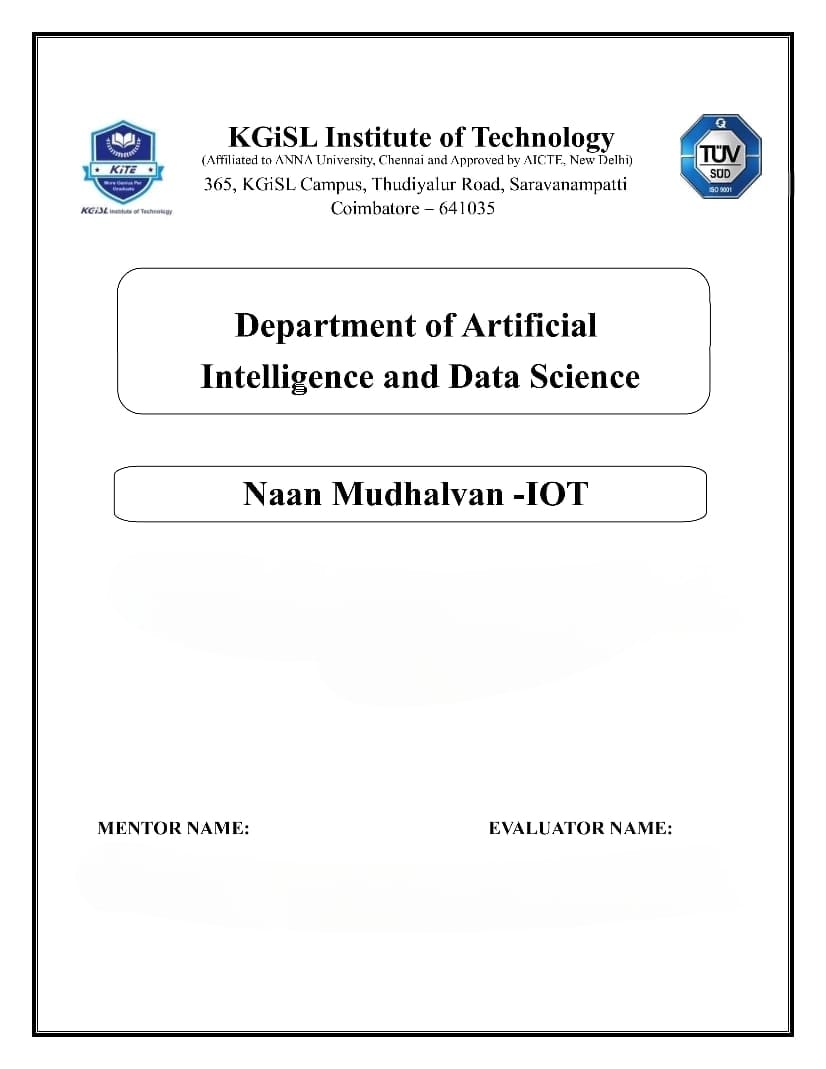
****

**PROBLEM STATEMENT : PUBLIC TRANSPORT OPTIMIZATION**

Mr.MOHANKUMAR.M

Ms.AKILANDESWARI.M

**Web-Based Real-Time Air Quality Monitoring System Components:**

**Frontend Development:**

User Interface (UI):

The UI is designed with a responsive layout, ensuring compatibility across various devices such as desktops, tablets, and mobile phones. The navigation is intuitively structured for seamless user interaction, featuring user dashboards that prominently display real-time air quality metrics.

Data Visualization:

To enhance user understanding, the platform employs diverse data visualization techniques. It includes charts, graphs, and numerical representations for both real-time and historical data. Additionally, interactive maps are incorporated to visually represent geographical variations in air quality.

User Interaction:

Interactive forms enable users to set preferences and thresholds. User-triggered actions, such as data downloads, are facilitated through buttons or toggles. Notifications are prominently displayed to alert users about relevant updates or alerts. Accessibility considerations adhere to standards, ensuring inclusivity for users with disabilities and those with diverse needs.

**Backend Development:**

Server-Side Logic:

The backend efficiently handles HTTP requests and responses, incorporating robust business logic for the processing and analysis of air quality data.

Database Integration:

Efficient storage of real-time and historical data is achieved through seamless integration with a suitable Database Management System (DBMS). Query optimization strategies are implemented to enhance data retrieval performance.

API Development:

RESTful API endpoints are established to facilitate communication with the frontend. Authentication and authorization mechanisms ensure secure data interaction.

**Web Development Technologies:**

Frontend Framework:

Dynamic UIs are constructed using frontend frameworks like React.js, Vue.js, or Angular.

Backend Framework:

Node.js, in conjunction with Express, Django, Flask, or a suitable alternative, serves as the backend framework.

Database:

Storage is facilitated by MongoDB, PostgreSQL, MySQL, or another database system based on project requirements.

Data Encryption:

Both data transmission and storage benefit from encryption, complemented by the implementation of secure authentication practices.

****

**Full Procedure:**

- The website will serve as a central hub for optimizing public transportation systems.

- Passengers can access real-time tracking and monitoring of public transport vehicles via an interactive map.

- The site offers a user-friendly trip planning tool, providing optimal routes and schedules.

- Comprehensive information on schedules, routes, and fare options is available to users.

- Real-time service updates will keep passengers informed about delays or disruptions.

- Passengers can explore the environmental benefits of public transport, including emissions reduction statistics.

- Helpful resources such as guides, FAQs, and passenger rights information are easily accessible.

- Users can provide feedback and report issues through a dedicated contact form.

- The website includes information about the project team's commitment to public transport improvement.

-Robust security measures are in place to protect user data and the site from cyber threats, and tools for collecting user feedback and analytics will drive continuous improvements.

**Header:**

- Logo and Project Name

- Display the project logo and name for brand identification.

Navigation Menu:

- Include a menu with links to different sections of the platform (e.g., Home, Real-Time Data, Historical Data, Health Recommendations).

User Authentication:

- If applicable, provide user authentication options (login, logout) and display the user's profile if logged in.

**Real-Time Air Quality Metrics:**

Air Quality Index (AQI):

- Prominently display the real-time AQI using standardized categories.

- Use color codes for quick visual recognition of air quality levels.

Pollutant Concentrations:

- Present real-time concentrations of specific pollutants (PM2.5, PM10, NO2, CO, O3, SO2, VOCs) through charts or numerical values.

Visual Representations:

- Include visual representations like gauges or progress bars to provide a quick overview of air quality conditions.

**Data Visualization:**

Graphs and Charts:

- Display historical trends of air quality parameters using line charts, bar graphs, or area charts.

- Allow users to customize the time range for historical data.

Interactive Maps:

- Showcase geographical variations in air quality through interactive maps.

- Allow users to zoom in and out, and click on specific locations for detailed information.

**User Interaction Section:**

- Preferences and Settings:

- Provide forms or settings options for users to customize preferences, such as notification thresholds or preferred units.

**Download Data:**

- Include a feature to download real-time or historical air quality data for research or personal use.

Notifications:

- Display notifications about changes in air quality or health risks.

**Footer:**

Contact Information:

- Provide contact details for support or inquiries.

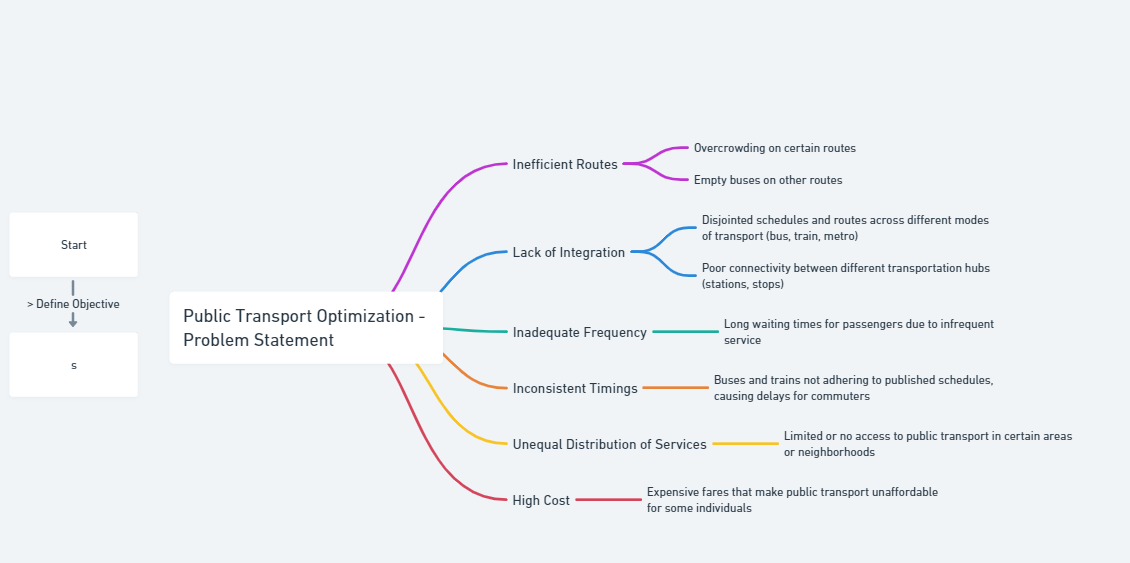
Links to Documentation:

- Include links to user manuals, API documentation, and other relevant resources

Privacy Policy and Terms of Use:

- Links to privacy policy and terms of use for transparency and legal compliance.

- Display loading indicators during data retrieval to inform users that the system is processing information.

****

**SOURCE CODE:**

import logging

import confluent\_kafka

from confluent\_kafka import Consumer

from confluent\_kafka.avro import AvroConsumer

from confluent\_kafka.avro.serializer import SerializerError

from tornado import gen

logger = logging.getLogger(\_\_name\_\_)

class KafkaConsumer:

"""Defines the base kafka consumer class"""

def \_\_init\_\_(

self,

topic\_name\_pattern,

message\_handler,

is\_avro=True,

offset\_earliest=False,

sleep\_secs=1.0,

consume\_timeout=0.1,

):

"""Creates a consumer object for asynchronous use"""

self.topic\_name\_pattern = topic\_name\_pattern

self.message\_handler = message\_handler

self.sleep\_secs = sleep\_secs

self.consume\_timeout = consume\_timeout

self.offset\_earliest = offset\_earliest

self.broker\_properties = {

'bootstrap.servers': 'PLAINTEXT://localhost:9094',

'default.topic.config': {'auto.offset.reset': 'earliest'},

'group.id': topic\_name\_pattern

}

if is\_avro is True:

self.broker\_properties["schema.registry.url"] = "http://localhost:8081"

self.consumer = AvroConsumer(self.broker\_properties)

else:

self.consumer = Consumer(self.broker\_properties)

pass

self.consumer.subscribe([topic\_name\_pattern],on\_assign=self.on\_assign)

def on\_assign(self, consumer, partitions):

"""Callback for when topic assignment takes place"""

for p in partitions:

consumer.seek(p)

logger.info(f"partitions assigned for {self.topic\_name\_pattern}")

consumer.assign(partitions)

async def consume(self):

"""Asynchronously consumes data from kafka topic"""

while True:

num\_results = 1

while num\_results > 0:

num\_results = self.\_consume()

await gen.sleep(self.sleep\_secs)

def \_consume(self):

"""Polls for a message. Returns 1 if a message was received, 0 otherwise"""

try:

msg = self.consumer.poll(timeout=1.0)

if msg is not None:

if msg.error() is not None:

self.message\_handler(msg)

return 1

else:

logger.error(msg.error())

return 0

else:

logger.debug("no message")

return 0

except SerializerError as error:

logger.error(f"Error consuming data: {error.message}")

return 0

def close(self):

self.consumer.close()

/\* Basic styling for the webpage \*/

body {

font-family: Arial, sans-serif;

}

header {

background-color: #0074D9;

color: #fff;

padding: 10px;

text-align: center;

}

main {

display: flex;

justify-content: space-around;

padding: 20px;

}

.map {

width: 60%;

border: 1px solid #ccc;

padding: 10px;

}

#map {

width: 100%;

height: 400px;

}

.info {

width: 35%;

border: 1px solid #ccc;

padding: 10px;

}

h2 {

font-size: 1.2em;

margin-bottom: 10px;

}