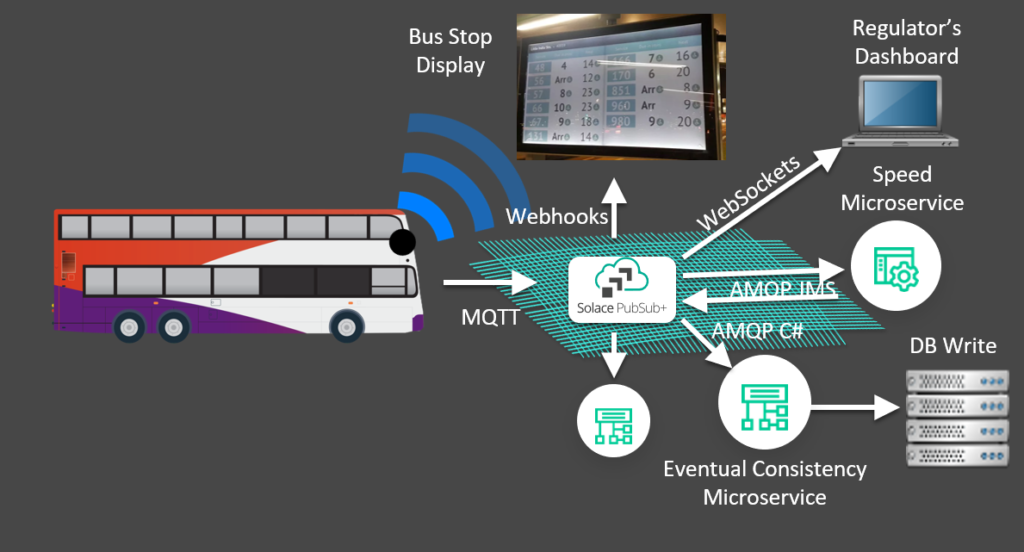
**PUBLIC TRANSPORT OPTIMIZATION USING IOT**

**PROJECT DESCRIPTION:**

By optimizing your public transport systems you can help citizens get where they’re going more quickly by reducing congestion on roadways and intelligently allocating and routing buses to areas with more travelers. That means locals, visitors and workers spend less time in transit and more time enjoying your city, successfully completing errands or getting to and from work. public transport may be the most important element of smart city planning because it affects everyone. All citizens and visitors need to get from one place to another, quickly and safely, and in today’s densely populated cities that means public transport.



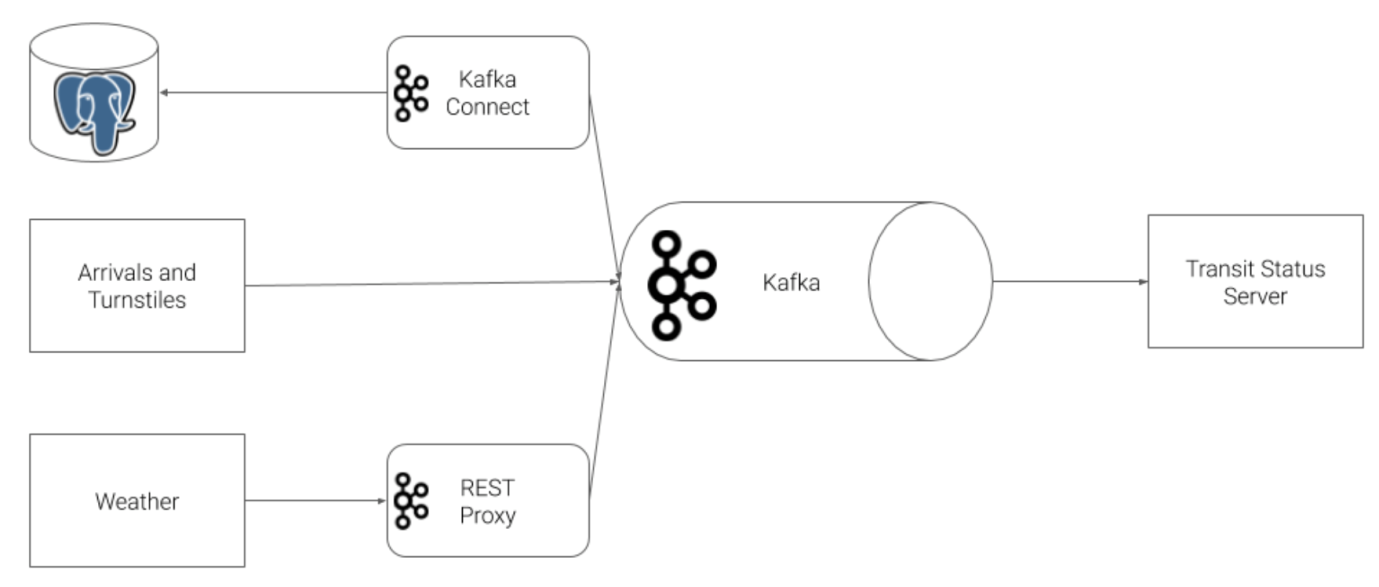
**IOT IMPLEMENTATION**

The project's core principle involves integrating IoT sensors into public transportation vehicles to revolutionize the efficiency and quality of public transit services. These sensors, including GPS and passenger counters, continuously gather real-time data. This data is then transmitted wirelessly to a central platform, where Python scripts process it. Passenger counts are aggregated, GPS data is analyzed for location and speed, and environmental conditions are assessed. Predictive algorithms can be deployed to estimate arrival times and detect anomalies. The processed data is made readily accessible to the public through user-friendly platforms like mobile apps or websites, enabling passengers to track vehicle locations and obtain estimated arrival times. This system ensures continuous monitoring and reporting, allowing transportation authorities to optimize services while maintaining a feedback loop for ongoing enhancements, ultimately improving the overall transit experience.

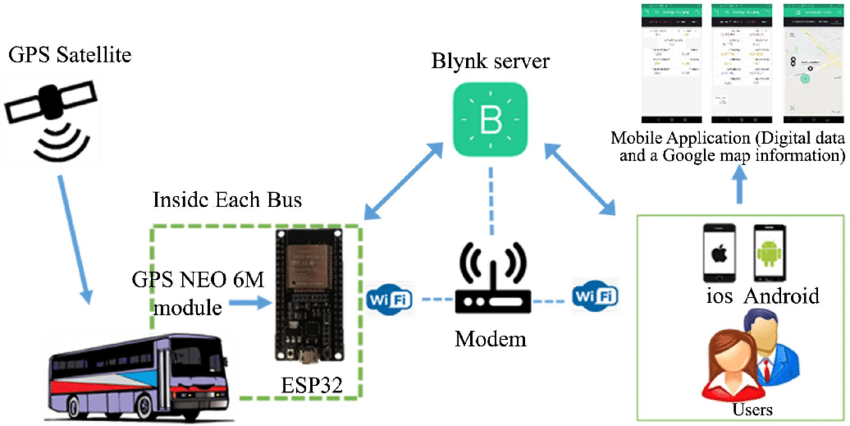
**METHODOLOGY**

In optimizing public transportation with IoT, gather real-time data on passenger flow, vehicle status, and traffic conditions. Analyze this data to enhance route planning, reduce delays, and improve overall efficiency. Implement IoT devices on vehicles for continuous monitoring and feedback loops, ensuring a dynamic and responsive transportation system. Public transportation optimization through IoT involves deploying sensors and devices to collect real-time data on factors like passenger numbers, vehicle status, and traffic conditions. This data is then analyzed to improve route planning, minimize delays, and enhance overall efficiency. By incorporating IoT devices on vehicles, a continuous feedback loop is established, allowing for dynamic adjustments and improved responsiveness in the transportation system.Use IoT sensors on vehicles for real-time data on passenger flow and traffic. Analyze to optimize routes, reduce delays, and enhance efficiency. Continuous monitoring ensures dynamic adjustments for improved responsiveness.

**BLOCK DIAGRAM**



**PROTOTYPE OF PUBLIC TRANSPORT OPTIMIZATION**

****

**APP DEVELOPMENT**

Build the app's frontend, implementing the designed UI and user interaction. Develop features such as route planning, real-time tracking, data visualization, and predictive algorithms.

**APP CODE:**

import logging

import confluent\_kafka

from confluent\_kafka import Consumer, OFFSET\_BEGINNING

from confluent\_kafka.avro import AvroConsumer,CachedSchemaRegistryClient

from confluent\_kafka.avro.serializer import SerializerError

from tornado import gen

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

class KafkaConsumer:

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

"""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

# Configure broker # Broker properties

self.broker\_properties = {

"bootstrap.servers" : "localhost:9092",

"group.id" : "my-consumer-group",}

if is\_avro is True:

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

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

else:

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

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

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

for partition in partitions:

partition.offset = OFFSET\_BEGINNING

logger.info("partitions assigned for %s", self.topic\_name\_pattern)

consumer.assign(partitions)

async def consume(self):

while True:

num\_results = 1

while num\_results > 0:

num\_results = self.\_consume()

await gen.sleep(self.sleep\_secs)

def \_consume(self):

message = self.consumer.poll(1.0)

if message is None:

logger.info("No message received by consumer.")

return 0

elif message.error() is not None:

logger.debug(f"error from consumer {message.error()}")

return 0

else:

try:

logger.info(message.value())

return 1

except KeyError as e:

logger.info(f"Failed to unpack message {e}")

return 0

def close(self):

self.consumer.close()

logger.info("Shutting down consumer.")

import logging

import faust

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

class Station(faust.Record):

stop\_id: int

direction\_id: str

stop\_name: str

station\_name: str

station\_descriptive\_name: str

station\_id: int

order: int

red: bool

blue: bool

green: bool

class TransformedStation(faust.Record):

station\_id: int

station\_name: str

order: int

line:

out\_topic = app.topic("faust.stations.transformed", partitions=1, value\_type=TransformedStation)

table = app.Table(

"stations.transformation.table",

default=int,

partitions=1,

changelog\_topic=out\_topic

@app.agent(topic)

async def StationProcess(stations):

async for station in stations:

transformed\_line = ""

if(station.red == True):

transformed\_line = "red"

elif(station.blue == True):

transformed\_line = "blue"

elif(station.green == True):

transformed\_line = "green"

else:

transformed\_line = "null"

transformed\_station = TransformedStation(

station\_id=station.station\_id,

station\_name=station.station\_name,

order=station.order,

line=transformed\_line

)

await out\_topic.send(value=transformed\_station)

if \_\_name\_\_ == "\_\_main\_\_":app.main()

def run\_server():

if topic\_check.topic\_exists("TURNSTILE\_SUMMARY") is False:

logger.fatal(

"Ensure that the KSQL Command has run successfully before running the web server!"

)

exit(1)

if topic\_check.topic\_pattern\_match("faust.stations.transformed") is False:

logger.fatal(

"Ensure that Faust Streaming is running successfully before running the web server!"

)

exit(1)

weather\_model = Weather()

lines = Lines()

application = tornado.web.Application(

[(r"/", MainHandler, {"weather": weather\_model, "lines": lines})]

)

application.listen(WEB\_SERVER\_PORT)

print("Building consumers....")

consumers = [

KafkaConsumer(

"weather",

weather\_model.process\_message,

offset\_earliest=True,

),

KafkaConsumer(

"faust.stations.transformed",

lines.process\_message,

offset\_earliest=True,

is\_avro=False,

),

**SIMULATION**

import datetime

import time

from enum import IntEnum

import logging

import logging.config

from pathlib import Path

import pandas as pd

# Import logging before models to ensure configuration is picked up

logging.config.fileConfig(f"{Path(\_\_file\_\_).parents[0]}/logging.ini")

from connector import configure\_connector

from models import Line, Weather

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

class TimeSimulation:

weekdays = IntEnum("weekdays", "mon tue wed thu fri sat sun", start=0)

ten\_min\_frequency = datetime.timedelta(minutes=10)

def \_\_init\_\_(self, sleep\_seconds=5, time\_step=None, schedule=None):

"""Initializes the time simulation"""

self.sleep\_seconds = sleep\_seconds

self.time\_step = time\_step

if self.time\_step is None:

self.time\_step = datetime.timedelta(minutes=self.sleep\_seconds)

# Read data from disk

self.raw\_df = pd.read\_csv(

f"{Path(\_\_file\_\_).parents[0]}/data/cta\_stations.csv"

).sort\_values("order")

# Define the train schedule (same for all trains)

self.schedule = schedule

if schedule is None:

self.schedule = {

TimeSimulation.weekdays.mon: {0: TimeSimulation.ten\_min\_frequency},

TimeSimulation.weekdays.tue: {0: TimeSimulation.ten\_min\_frequency},

TimeSimulation.weekdays.wed: {0: TimeSimulation.ten\_min\_frequency},

TimeSimulation.weekdays.thu: {0: TimeSimulation.ten\_min\_frequency},

TimeSimulation.weekdays.fri: {0: TimeSimulation.ten\_min\_frequency},

TimeSimulation.weekdays.sat: {0: TimeSimulation.ten\_min\_frequency},

TimeSimulation.weekdays.sun: {0: TimeSimulation.ten\_min\_frequency},

}

self.train\_lines = [

Line(Line.colors.blue, self.raw\_df[self.raw\_df["blue"]]),

Line(Line.colors.red, self.raw\_df[self.raw\_df["red"]]),

Line(Line.colors.green, self.raw\_df[self.raw\_df["green"]]),

]

def run(self):

curr\_time = datetime.datetime.utcnow().replace(

hour=0, minute=0, second=0, microsecond=0

)

logger.info("Beginning simulation, press Ctrl+C to exit at any time")

logger.info("loading kafka connect jdbc source connector")

configure\_connector()

logger.info("beginning cta train simulation")

weather = Weather(curr\_time.month)

try:

while True:

logger.debug("simulation running: %s", curr\_time.isoformat())

# Send weather on the top of the hour

if curr\_time.minute == 0:

weather.run(curr\_time.month)

\_ = [line.run(curr\_time, self.time\_step) for line in self.train\_lines]

curr\_time = curr\_time + self.time\_step

time.sleep(self.sleep\_seconds)

except KeyboardInterrupt as e:

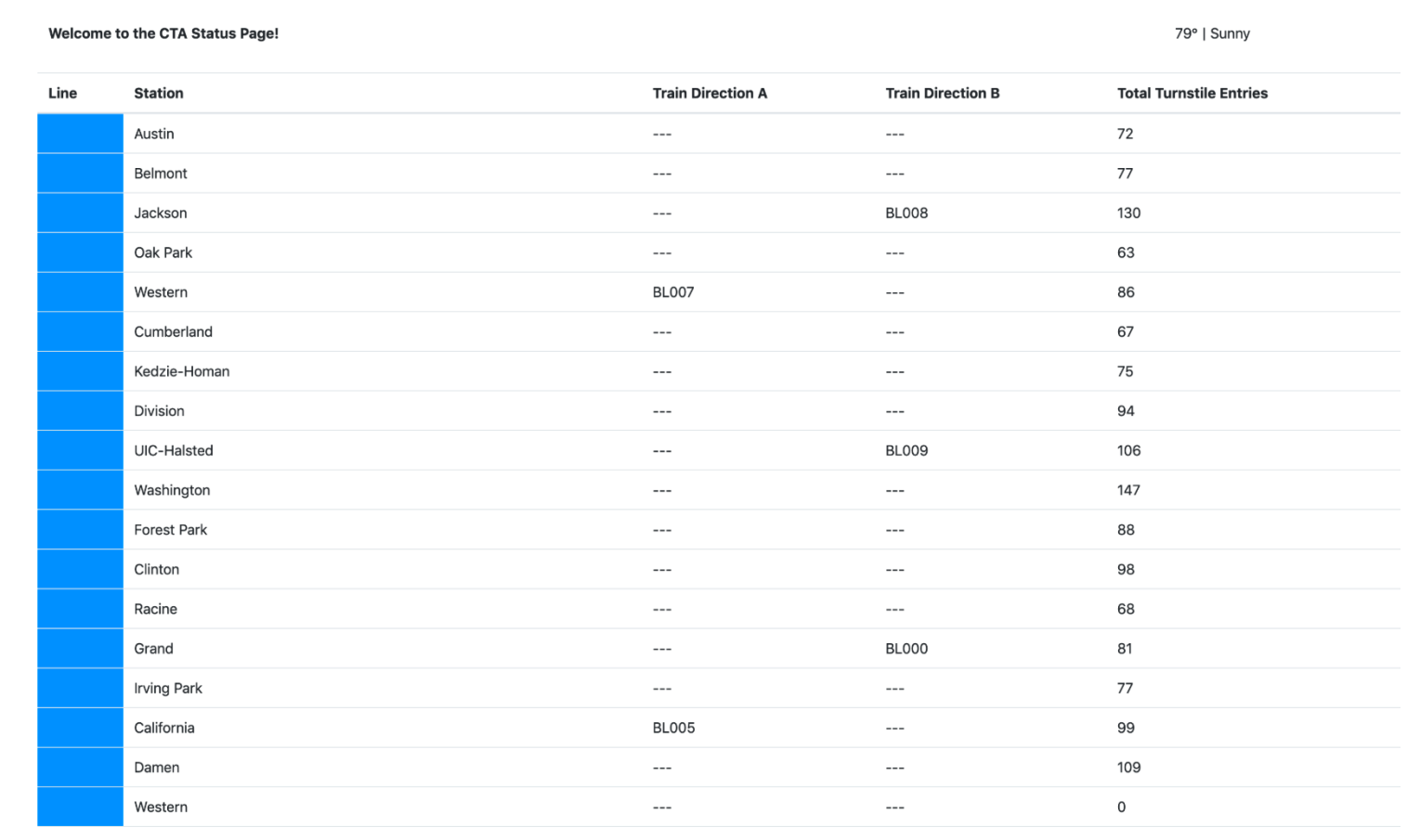
logger.info("Shutting down")= [line.close() for line in self.train\_lines]

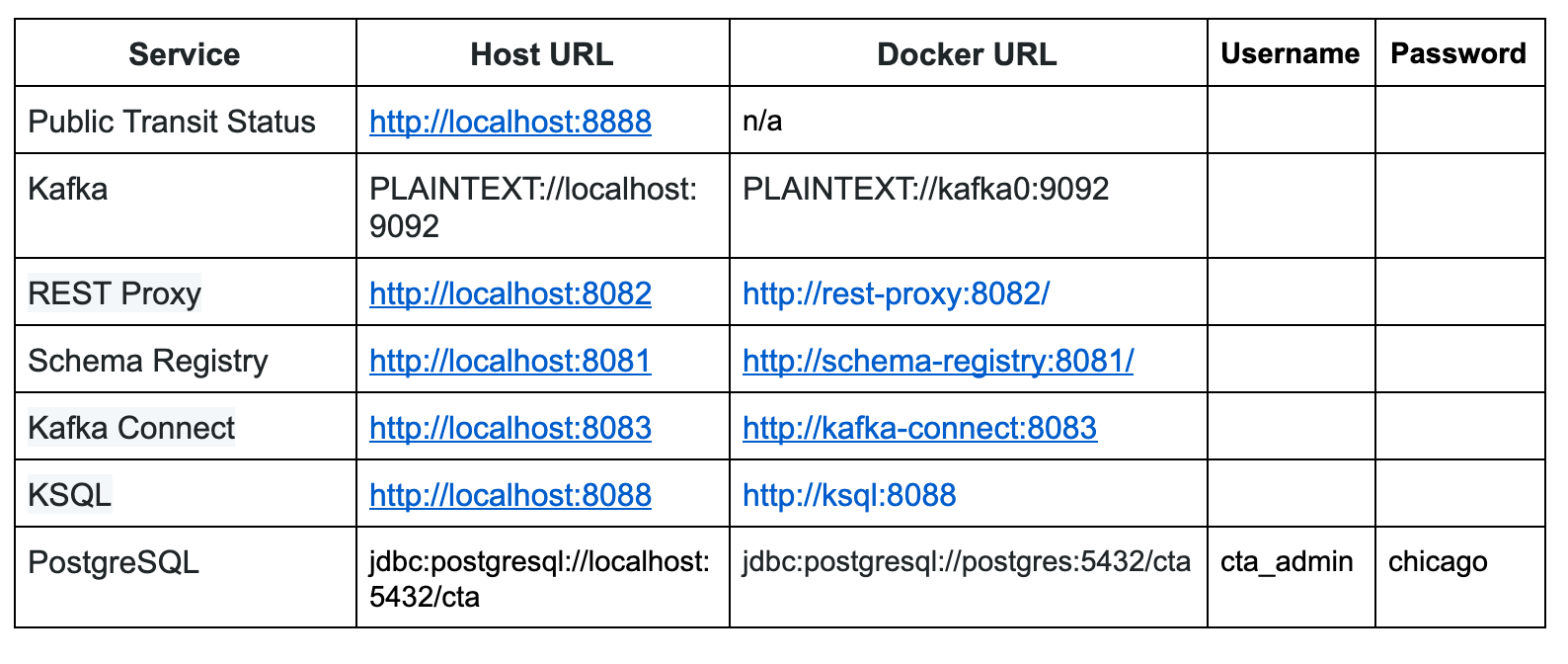
if \_\_name\_\_ == "\_\_main\_\_": TimeSimulation().run()

**WORKING PRINCIPLE OF THE PROTOTYPE**

The prototype utilizes IoT sensors on vehicles to capture real-time data such as passenger numbers and traffic conditions. This data is then analyzed to optimize transportation routes, reduce delays, and enhance overall efficiency. The continuous monitoring through IoT enables dynamic adjustments, ensuring the system responds effectively to changing variables in public transportation.

**DATA ANALYSIS FROM THE PROTOTYPE**





**CONCLUSION**

The project to integrate IoT sensors into public transportation vehicles for monitoring ridership, tracking locations, and predicting arrival times is a transformative initiative aimed at enhancing the efficiency and quality of public transportation services. By leveraging real-time data from IoT sensors, the system provides passengers with valuable information, such as accurate vehicle locations and estimated arrival times, empowering them to make informed travel decisions. Simultaneously, transportation authorities benefit from improved operational insights, enabling better resource allocation and service optimization. The integration of predictive algorithms and user-friendly interfaces ensures that the project aligns with modern technological advancements, making public transit more reliable and convenient for everyone. As an ongoing effort, the project sets the stage for continuous improvement and innovation in the realm of public transportation, ultimately contributing to more sustainable and efficient urban mobility solutions.