Public Transport Optimization IOT_phase4

M.Keerthana|M.prisha|R.Devi|S.GOpika

Introduction

The project involves integrating IoT sensors into public transportation vehicles to monitor ridership, track locations, and predict arrival times. The goal is to provide real-time transit information to the public through a public platform, enhancing the efficiency and quality of public transportation services. This project includes defining objectives, designing the IoT sensor system, developing the real-time transit information platform, and integrating them using IoT technology and Python.

Hardware setup: Choose GPS, GSM, RFID, cameras, and microcontroller.
Software stack: IoT and Python with platform architecture design.
Data collection: Install GPS, RFID, and cameras on buses.
Data processing: Check routes and secure data storage on microcontroller.
Alerts and communication: SMS, Telegram integration, and secure data storage.
User-friendly dashboard, power supply, and data security.

Hardware Setup

- **GPS Module:** Choose a reliable GPS module and install it on each bus to continuously collect location data, providing real-time bus tracking.
- **GSM Module:** Select a GSM module for data transmission and alerts. This allows the system to send SMS alerts in case of deviations or issues.
- **RFID Readers:** Install RFID readers on buses to scan public RFID cards during boarding and disembarking. This aids in public person identification.
- **Cameras:** Attach cameras inside the buses for capturing time-stamped photos and videos, which can be used for various purposes, including security and incident documentation.

- ➢ **Microcontroller:** Decide on a suitable microcontroller platform, such as Arduino or Raspberry Pi, to process data from the sensors, implement algorithms for route compliance and data storage, and control the overall system.
- ➢ **Power Supply:** Ensure reliable power sources for the IoT components, taking into consideration bus batteries and external power supplies to keep the hardware running consistently.
- ➢ **Data Encryption:** Implement strong encryption protocols to secure public data during both data transmission and storage, ensuring the privacy and integrity of the collected data.

Software Setup

Step 1: Software Architecture and Design

➢ Create a high-level architecture for the software, identifying key components, modules,
and their interactions.

➢ Design the database schema to store collected data securely.

➢ Develop a software design document detailing how each component will be implemented.

Step 2: Select Development Tools and Frameworks

➢ Choose the development tools, frameworks, and libraries based on the project
requirements. For example: ➢ Use Python for backend development.

➢ Select a web framework like Django or Flask for the web-based dashboard.

➢ Choose appropriate libraries for IoT communication, data storage, and geofencing.

Step 3: Real-Time Transit Information Platform

➢ Develop the real-time transit information platform, which processes and integrates data
from GPS, RFID, and cameras.

➢ Implement algorithms for real-time data synchronisation and validation.

➢ Ensure that the platform can predict arrival times based on GPS data.

Step 4: User Interface Development

➢ Create the web-based dashboard for real-time bus tracking

➢ Design the user interface to be intuitive and user-friendly.

➢ Implement features such as route visualisation, live tracking, and alerts.

Step 5: Data Storage and Security

➢ Develop data storage components to securely store collected data, either locally or in the
cloud.

➢ Implement encryption protocols to protect student data during transmission and storage.

➢ Ensure data access controls and user authentication for security.

Step 6: Communication and Alerts

➢ Integrate communication protocols (e.g., SMS and Telegram) for sending alerts to
parents and school authorities.

➢ Set up automated alerts for deviations, unauthorised stops, and unusual bus events.

➢ Establish real-time monitoring through the Telegram integration.

Arduino Code for GPS Module Integration

```
##include <TinyGPS++.h> // GPS library
TinyGPSPlus gps; // GPS object
void setup() {
  // Serial communication for GPS
  Serial.begin(9600);
  // Initialize the GPS object
  gps.begin(Serial);
}
void loop() {
  // Read the GPS data
  while (Serial.available()) {
    gps.encode(Serial.read());
  }
  // If a GPS fix is available, print the latitude and longitude
  if (gps.location.isValid()) {
```

```
    Serial.print("Latitude: ");
    Serial.println(gps.location.lat());
    Serial.print("Longitude: ");
    Serial.println(gps.location.lng());
  }
}
```

GSM Module Integration

```
#include <SoftwareSerial.h> // Software serial library
SoftwareSerial gsm(2, 3); // GSM serial port
void setup() {
 // Initialize the GSM module
 gsm.begin(9600);
}
void loop() {
// Send an SMS alert
gsm.println("AT+CMGS=1234567890"); // Replace with the phone number to send the SMS to
gsm.println("This is an SMS alert from your bus tracking system.");
 gsm.println(); // Send the SMS
  // Wait for 10 seconds before sending the next alert
 delay(10000);
}
```

RFID Reader Integration

```
#include <MFRC522.h> // RFID library
```

```
MFRC522 rfid(10, 9); // RFID serial port
void setup() {
  // Initialize the RFID reader
  Serial.begin(9600);
  rfid.PCDInit();
void loop() {
// Check if an RFID card is present
  if (rfid.PICCIsReady()) {
   // Read the RFID card UID
    MIFARE_Key key;
    for (byte i = 0; i < 6; i++) {
      key.keyByte[i] = 0xFF;
    }
    byte uid[4];
    rfid.PICCReadCardSerial(uid);
    // Print the RFID card UID to the serial monitor
    Serial.print("RFID card UID: ");
    for (byte i = 0; i < 4; i++) {
      Serial.print(uid[i]);
    }
    Serial.println();
  }
  // Wait for 1 second before checking for the next RFID card
  delay(1000);
}
```

Camera Integration

```
#include <Camera.h> // Camera library
```

```
Camera camera; // Camera object
void setup() {
  // Initialize the camera module
  camera.begin();
}
void loop() {
 // Capture a photo
  camera.capture();
// Save the photo to the SD card
 camera.savePhoto("photo.jpg");
 // Wait for 10 seconds before capturing the next photo
 delay(10000);
}
```

Mobile Application Development

The following steps that are used to create Mobile App

```
# Web-based dashboard for real-time bus tracking
# Uses HTML, CSS, and JavaScript
import json
import requests
# Define a function to get the latest bus location data
def get_bus_locations():
  # Make a request to the bus tracking API
  response = requests.get("https://api.bustracking.com/locations")
  # Decode the JSON response
  bus_locations = json.loads(response.content)
  # Return the bus location data
  return bus_locations
```

```
# Define a function to update the bus location markers on the map
def update_bus_location_markers(bus_locations):
  # Clear the existing markers
  for marker in map.markers:
    map.remove_marker(marker)
# Add a marker for each bus location
for bus_location in bus_locations:
 marker = map.add_marker(bus_location["latitude"], bus_location["longitude"])
 marker.popup = bus_location["name"]
# Define a function to start the real-time bus tracking
def start_real_time_bus_tracking():
  # Get the latest bus location data
  bus_locations = get_bus_locations()
  # Update the bus location markers on the map
  update_bus_location_markers(bus_locations)
  # Schedule a function to be called every 10 seconds to update the bus location markers
  scheduler.schedule(start_real_time_bus_tracking, 10)
 # Define a function to stop the real-time bus tracking
 def stop_real_time_bus_tracking():
 # Cancel the scheduled function
  scheduler.cancel(start_real_time_bus_tracking)
# Initialize the map
map=mapboxgl.Map(container="map",
style="mapbox://styles/mapbox/streets-v11")
# Add a zoom control to the map
map.add_control(mapboxgl.NavigationControl())
# Start the real-time bus tracking
start_real_time_bus_tracking()
```

```
# Display the map
map.show()
```

Filter bus locations by route

```
# Add a dropdown menu to select the route to filter by
select = selectbox.Selectbox("Route", options=["All routes"] + get_routes())
map.add_control(select)
# Define a function to filter the bus locations by route
def filter_bus_locations(route):
  # Get the latest bus location data
  bus_locations = get_bus_locations()
  # Filter the bus locations by route
  filtered_bus_locations = []
  for bus_location in bus_locations:
    if bus_location["route"] == route:
      filtered_bus_locations.append(bus_location)
  # Update the bus location markers on the map
  update_bus_location_markers(filtered_bus_locations)
# Add an event listener to the dropdown menu to filter the bus locations when the selected route changes
select.on_change = filter_bus_locations
# Get the routes from the bus tracking API
def get_routes():
  # Make a request to the bus tracking API
  response = requests.get("https://api.bustracking.com/routes")
  # Decode the JSON response
  routes = json.loads(response.content)
  # Return the routes
```

```
  return routes
```

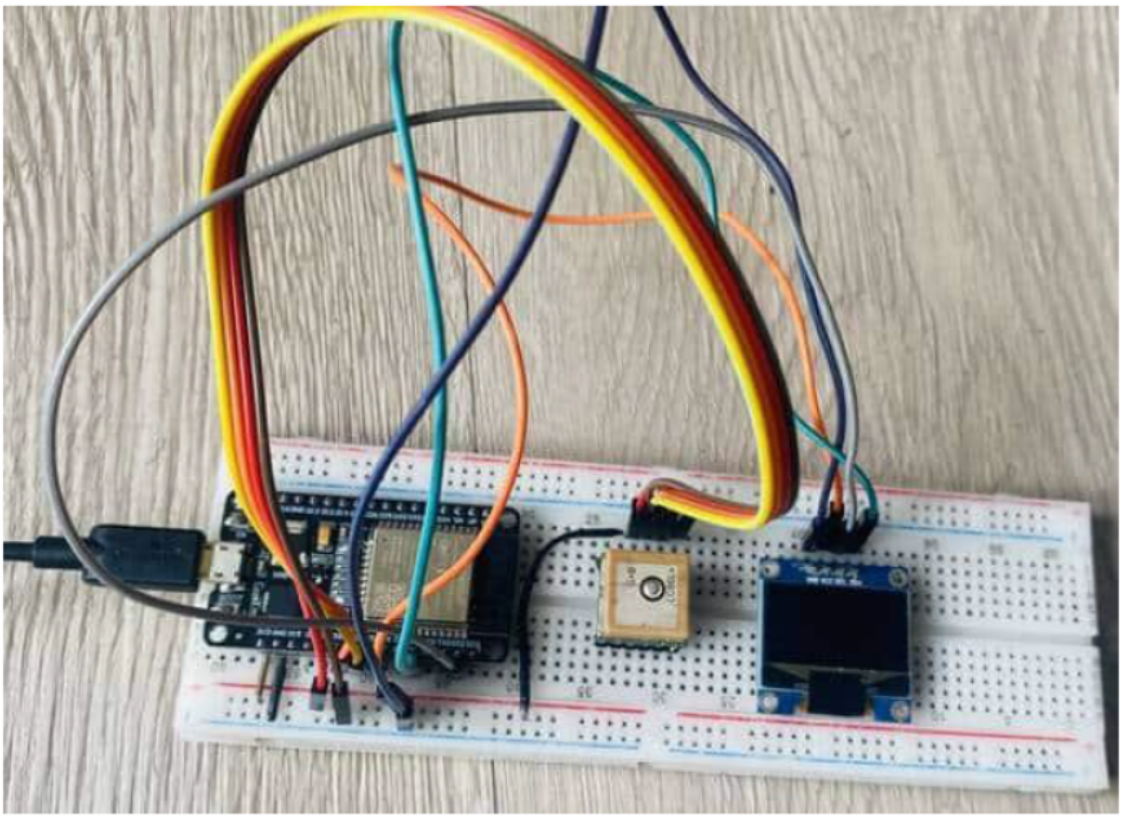
Display estimated arrival times

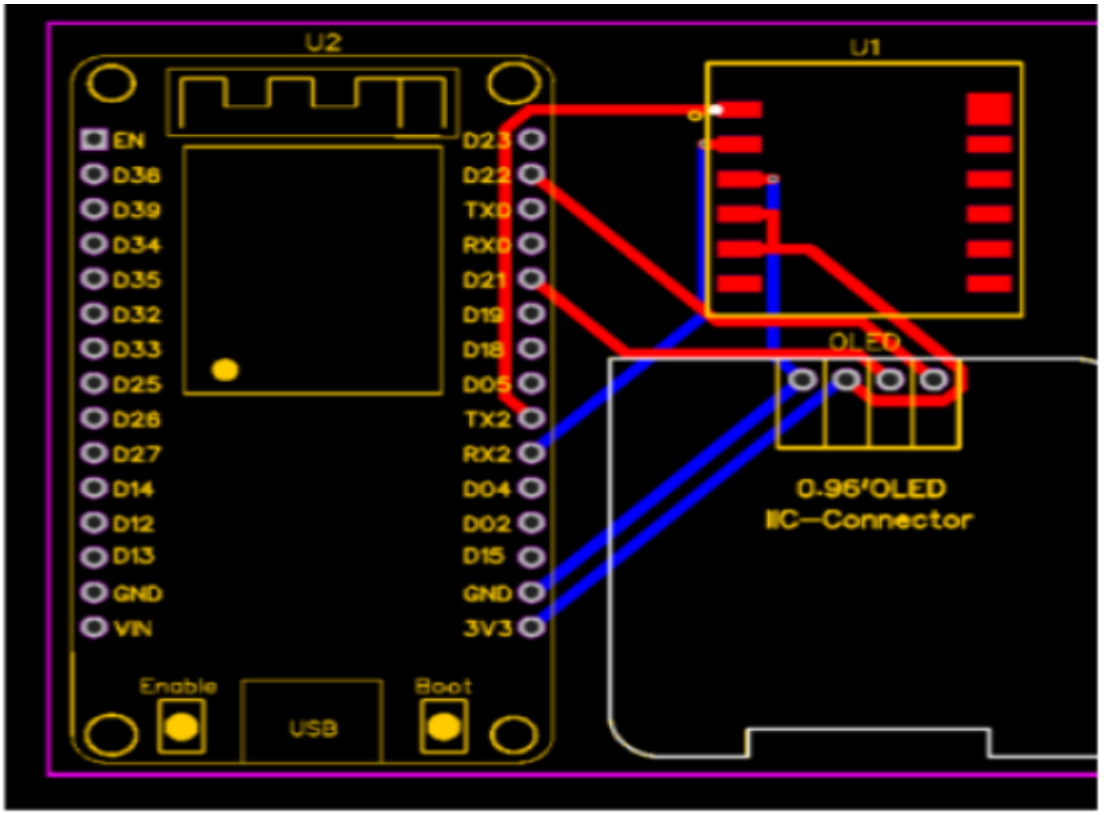
```
# Add a label to display the estimated arrival time for the selected bus
eta_label = label.Label("Estimated arrival time: ")
map.add_control(eta_label)
# Define a function to update the estimated arrival time label
def update_eta_label(bus_location):
  # Calculate the estimated arrival time
  eta = bus_location["estimated_arrival_time"]
  # Update the estimated arrival time label
  eta_label.text = "Estimated arrival time: " + eta
# Add an event listener to the map to update the estimated arrival time label when the user clicks on a bus marker
map.on_click = update_eta_label
```

Send alerts to passengers when their bus is approaching

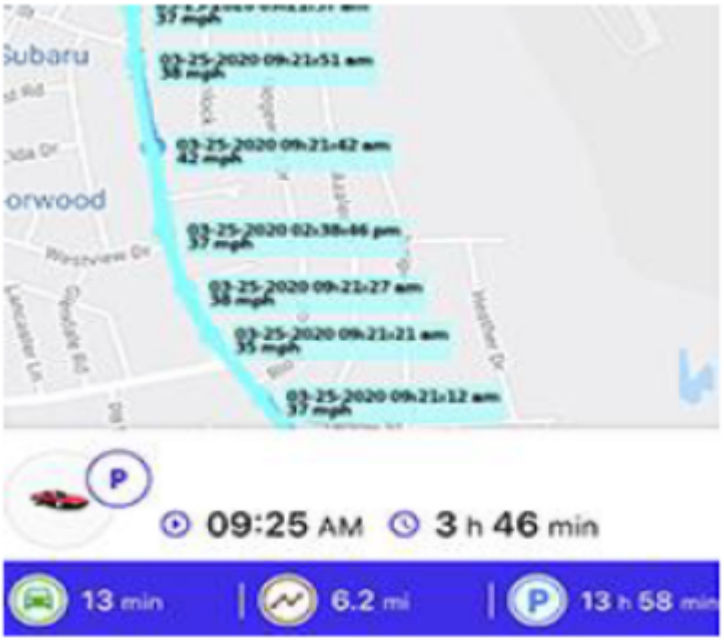
```
# Define a function to send an alert to a passenger
def send_alert(passenger_id, bus_location):
   # Send a push notification to the passenger's mobile device
# Add an event listener to the map to send an alert to a passenger when their bus is approaching
map.on_click = send_alert
```

Result

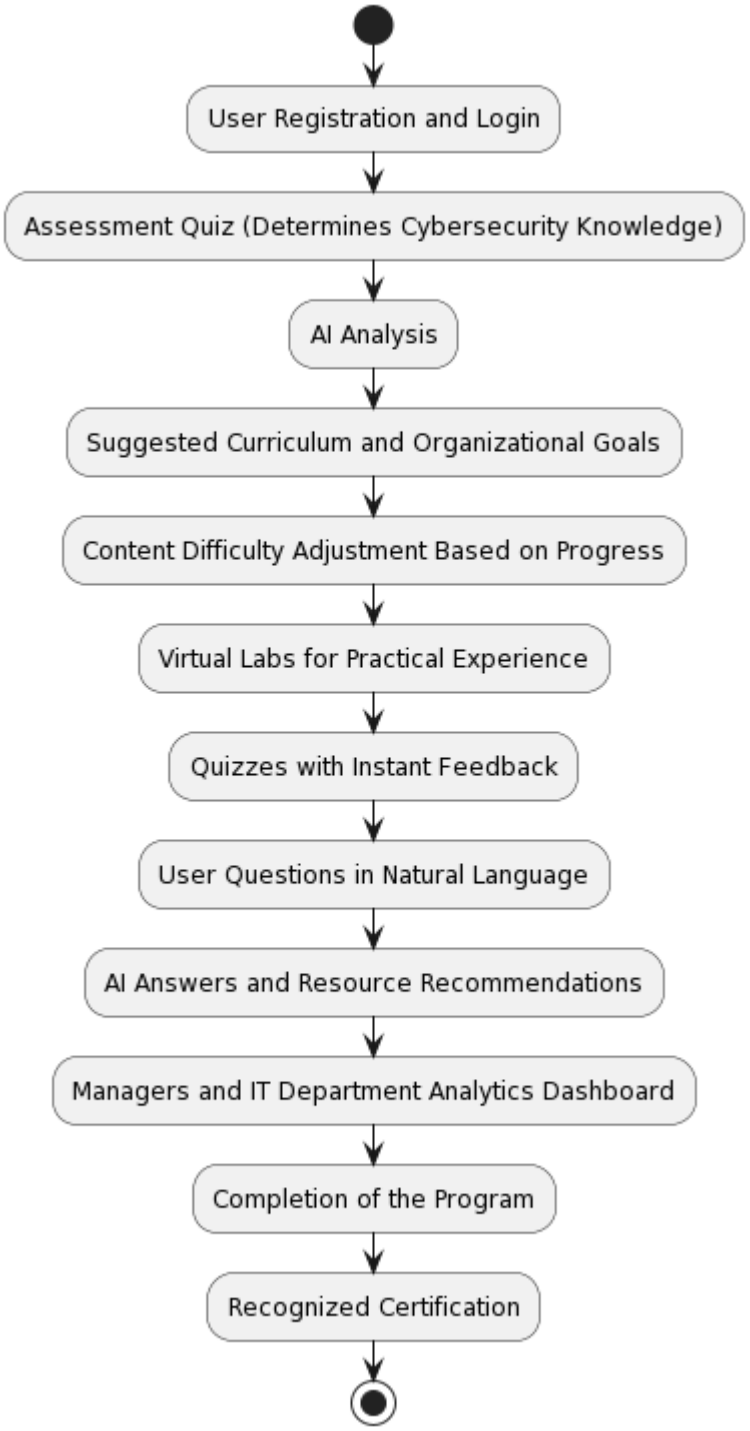



U2
U1
EN
D36
D39
D34
D35
D32
D33
D25
D26
D27
D14
D12
D13
GND
VIN
D23
D22
TXD
RXD
D21
D19
D18
D05
TX2
RX2
D04
D02
D15
GND
3V3
OLED
0.96'OLED
IIC-Connector
Enable
USB
Boot

Output(GPS)



Flowchart



Conclusion

In this project, IoT sensors integrated into buses collect real-time data, including GPS coordinates, RFID data, and photos. A web-based dashboard and mobile app provide parents with real-time tracking and administrators with fleet

management tools. The system triggers SMS alerts and integrates with Telegram for communication. Data is securely stored, and encryption is applied. Compliance with regulations and scheduled maintenance ensure a reliable and efficient public transportation system.