SMART BIKE RFID ENABLED ENGIENE SELF START & LOCATION TRACKING

MINI PROJECT REPORT

Submitted by

NITHILAN M	**2116230701216**
JAYASUDHAN V	**2116230701131**

In partial fulfillment for the award of the degree

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE

ANNA UNIVERSITY: CHENNAI 600 025

MAY 2025

CHAPTER 6

SAMPLE CODING

Arduino RFID Program

```
#include <SPI.h>
#include <MFRC522.h>
#define RST_PIN 9
#define SS_PIN 10
#define RELAY_PIN 8
MFRC522 rfid(SS_PIN, RST_PIN);
bool motorState = false;
String authorizedUIDs[] = {"71CA48C", "33D365C8"};
void setup() {
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();
  pinMode(RELAY_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, LOW);
  Serial.print("Place your RFID tag near the reader.");
}
void loop() {
  if (!rfid.PICC_IsNewCardPresent() || !rfid.PICC_ReadCardSerial()) {
    return;
  }
  String uid = getUID();
  Serial.println("Scanned UID: " + uid);
  if (isAuthorizedTag(uid)) {
  toggleMotor();
  } else {
    Serial.println("Unauthorized tag!");
  }
  rfid.PICC_HaltA();
}

void toggleMotor() {
  motorState = !motorState;
  digitalWrite(RELAY_PIN, motorState ? HIGH : LOW);
  if (motorState) {
    Serial.println("Motor turned OFF");
  } else {
    Serial.println("Motor turned ON");
  }
}
String getUID() {
  String uid = "";
  for (byte i = 0; i < rfid.uid.size; i++) {
    uid += String(rfid.uid.uidByte[i], HEX);
  }
  uid.toUpperCase();
  return uid;
}
bool isAuthorizedTag(String uid) {
  for (String authorizedUID : authorizedUIDs) {
    if (uid == authorizedUID) {
      return true;
    }
  }
  return false;
}
```

ESP32 GPS Program

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <TinyGPSPlus.h>
#include <HardwareSerial.h>

const char* ssid = "OnePlus 11R 5G";
const char* password = "i8r23pg3";
const char* supabase_url =
"https://oajubsmkcazrrflpzauf.supabase.co/rest/v1/gps_data";



const char* supabase_api_key =
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdXBhYmFzZSIs
InJlZiI6Im9hanVic21rY2F6cnJmbHB6YXVmIiwicm9sZSI6ImFub24iLCJp
YXQiOjE3NDQ0NzU3NjAsImV4cCI6MjA2MDA1MTc2MH0.G3LMSrw
KEB1rO7hGVv60iAVvt_ptcjAv26ecN-LwXk0";

HardwareSerial gpsSerial(1);
TinyGPSPlus gps;

void setup() {
  Serial.begin(9600);
  gpsSerial.begin(9600, SERIAL_8N1, 16, 17);
  Serial.print("Connecting to WiFi");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected!");
}
void loop() {
  while (gpsSerial.available() > 0) {
    gps.encode(gpsSerial.read());
  }
  if (gps.location.isUpdated()) {
    double latitude = gps.location.lat();
    double longitude = gps.location.lng();
    String maps_url = "https://maps.google.com/?q=" + String(latitude, 6) +
"," + String(longitude, 6);
    Serial.println("Sending GPS data to Supabase...");
    Serial.println("Lat: " + String(latitude, 6));
    Serial.println("Lng: " + String(longitude, 6));
    Serial.println("URL: " + maps_url);
    if (WiFi.status() == WL_CONNECTED) {
      HTTPClient http;
      http.begin(supabase_url);
      http.addHeader("Content-Type", "application/json");
      http.addHeader("apikey", supabase_api_key);
      http.addHeader("Authorization", "Bearer " + String(supabase_api_key));
      String payload = "{";
      payload += "\"latitude\": " + String(latitude, 6) + ",";
      payload += "\"longitude\": " + String(longitude, 6) + ",";
      payload += "\"maps_url\": \"" + maps_url + "\"";
      payload += "}";
      int httpResponseCode = http.POST(payload);
      Serial.print("HTTP Response code: ");
      Serial.println(httpResponseCode);
      http.end();
    } else {
      Serial.println("WiFi not connected");
    }
    delay(10000);
  }
}
```

Web Application for GPS Tracking Dashboard

1. LiveTracking.tsx

```
import React, { useEffect, useState } from 'react';
import { LocationData } from '@/lib/types';
import { Button } from '@/components/ui/button';
import { Card, CardContent, CardFooter, CardHeader, CardTitle } from
'@/components/ui/card';
import { MapPin, Clock } from 'lucide-react';
import { supabase } from '@/integrations/supabase/client';
import { useToast } from '@/hooks/use-toast';
import { Separator } from '@/components/ui/separator';
import Map from './Map';

interface LiveTrackingProps {
  initialLocation?: LocationData | null;
  onClose: () => void;
}

const LiveTracking = ({ initialLocation, onClose }: LiveTrackingProps) => {
const [liveLocation, setLiveLocation] = useState<LocationData |
null>(initialLocation || null);
  const [isConnected, setIsConnected] = useState(true);
  const [lastUpdate, setLastUpdate] = useState<Date | null>(
    initialLocation ? new Date(initialLocation.inserted_at) : null
  );
  const { toast } = useToast();
```

```
useEffect(() => {
  console.log('Setting up realtime subscription');
  const channel = supabase
    .channel('gps_updates')
    .on(
      'postgres_changes',
      { event: 'INSERT', schema: 'public', table: 'gps_data' },
      (payload) => {
        console.log('New GPS data received:', payload);
        const newLocation = {
          id: payload.new.id as string,
          latitude: payload.new.latitude as number,
          longitude: payload.new.longitude as number,
          maps_url: payload.new.maps_url as string | undefined,
          inserted_at: new Date(payload.new.inserted_at).toISOString()
        } as LocationData;

        setLiveLocation(newLocation);
        setLastUpdate(new Date(newLocation.inserted_at));
        setIsConnected(true);

        toast({
          title: "Live Update",
          description: "Received new GPS position",
          duration: 3000,
        });
      }
    )
    .subscribe((status) => {
      console.log('Subscription status:', status);
    });

  return () => {
    console.log('Cleaning up subscription');
    supabase.removeChannel(channel);
  };
}, [toast]);

useEffect(() => {
  if (!lastUpdate) return;

  const interval = setInterval(() => {
    const now = new Date();
    const diffSeconds = (now.getTime() - lastUpdate.getTime()) / 1000;

    if (diffSeconds > 30) {
      setIsConnected(false);
    }
  }, 5000);

  return () => clearInterval(interval);
}, [lastUpdate]);

const getGoogleMapsLink = (lat: number, lng: number) => {
  return `https://www.google.com/maps?q=${lat},${lng}`;
};
```

```
 return (
   <div className="fixed inset-0 bg-background z-50 flex flex-col">
     <header className="bg-white shadow-sm border-b p-4 flex justify-between
items-center">
       <div className="flex items-center">
         <MapPin className="text-tracking-primary mr-2 h-5 w-5" />
         <h2 className="text-xl font-semibold">Live GPS Tracking</h2>
         <div className={`ml-3 h-2.5 w-2.5 rounded-full ${isConnected ?
'bg-green-500' : 'bg-red-500'}`}></div>
         <span className="ml-1 text-sm text-gray-500">
           {isConnected ? 'Connected' : 'Waiting for updates...'}
         </span>
       </div>
       <Button variant="ghost" size="sm" onClick={onClose}>
         Close
       </Button>
     </header>

     <div className="flex-1 flex flex-col md:flex-row">
       <div className="flex-1 relative">
         <Map
           locations={liveLocation ? [liveLocation] : []}
           selectedLocation={liveLocation}
           onSelectLocation={() => {}}
           isLoading={false}
         />
       </div>

       <div className="w-full md:w-80 bg-white border-l overflow-auto p-4">
```

```
          <h3 className="font-semibold mb-2">Latest Position</h3>

          {liveLocation ? (
            <Card className="bg-gray-50">
              <CardHeader className="pb-2">
                <CardTitle className="text-base flex items-center">
                  <MapPin className="h-4 w-4 mr-1 text-tracking-primary" />
                  GPS Location
                </CardTitle>
              </CardHeader>
              <CardContent className="space-y-4">
                <div className="grid grid-cols-2 gap-2 text-sm">
                  <div>
                    <span className="text-gray-500">Latitude:</span>
{liveLocation.latitude.toFixed(6)}
                  </div>
                  <div>
                    <span className="text-gray-500">Longitude:</span>
{liveLocation.longitude.toFixed(6)}
                  </div>
                </div>

                <Separator />

                <div className="flex justify-between text-xs">
                  <div className="flex items-center gap-1">
                    <Clock className="h-3 w-3" />
                    <span>{new
Date(liveLocation.inserted_at).toLocaleTimeString()}</span>
```

```
                </div>
                <div>
                  <span>{new
Date(liveLocation.inserted_at).toLocaleDateString()}</span>
                </div>
              </div>
            </CardContent>
            <CardFooter>
              <Button
                variant="outline"
                size="sm"
                className="w-full text-xs"
                onClick={() => {
                  const mapUrl = liveLocation.maps_url ||
                    getGoogleMapsLink(liveLocation.latitude,
liveLocation.longitude);
                  window.open(mapUrl, '_blank');
                }}
              >
                Open in Google Maps
              </Button>
            </CardFooter>
          </Card>
        ) : (
          <div className="text-center p-8 text-gray-500">
            <p>Waiting for GPS updates...</p>
            <p className="text-xs mt-2">No live location data received yet</p>
          </div>
        )}
```

```
          <div className="mt-4">
            <h4 className="font-medium mb-2">About Live Tracking</h4>
            <p className="text-sm text-gray-600">
              This view automatically updates when new GPS data is received.
              Each new location will be displayed on the map immediately.
            </p>
          </div>
        </div>
      </div>
    </div>
  );
};

export default LiveTracking;
```

2. LocationSidebar.tsx

```
import React from 'react';
import { LocationData } from '@/lib/types';
import { ScrollArea } from '@/components/ui/scroll-area';
import { Button } from '@/components/ui/button';
import { MapPin, Clock, Calendar, ExternalLink, AlertCircle } from
'lucide-react';
import { Alert, AlertDescription, AlertTitle } from '@/components/ui/alert';

interface LocationSidebarProps {
  locations: LocationData[];
  selectedLocation: LocationData | null;
  onSelectLocation: (location: LocationData) => void;
  isLoading: boolean;
}

const LocationSidebar = ({
  locations,
  selectedLocation,
  onSelectLocation,
  isLoading
}: LocationSidebarProps) => {

  if (isLoading) {
    return (
      <div className="w-full h-full flex items-center justify-center">
        <div className="animate-pulse flex flex-col items-center">
          <div className="h-4 bg-slate-200 rounded w-3/4 mb-2.5"></div>
          <div className="h-4 bg-slate-200 rounded w-1/2"></div>
        </div>
      </div>
    );
  }

  if (locations.length === 0) {
    return (
      <div className="w-full h-full flex flex-col items-center justify-center
p-4">
        <Alert variant="destructive" className="mb-4">
          <AlertCircle className="h-4 w-4" />
          <AlertTitle>No Data Found</AlertTitle>
          <AlertDescription>
            No GPS location data was found in the database.
          </AlertDescription>
        </Alert>
        <p className="text-sm text-gray-500 text-center mt-2">
          Make sure your device is sending data to the Supabase database.
        </p>
      </div>
    );
  }

  const sortedLocations = [...locations].sort((a, b) =>
    new Date(b.inserted_at).getTime() - new Date(a.inserted_at).getTime()
  );

  return (
    <ScrollArea className="h-full">
      <div className="p-4">
        <h3 className="text-lg font-semibold mb-4">GPS History
({sortedLocations.length})</h3>

        <div className="space-y-4">
          {sortedLocations.map((location) => {
            const isSelected = selectedLocation?.id === location.id;
            const time = new Date(location.inserted_at).toLocaleTimeString();
            const date = new Date(location.inserted_at).toLocaleDateString();
            const mapUrl = location.maps_url ||

`https://www.google.com/maps?q=${location.latitude},${location.longitude}`;
```

```
return (
  <div
    key={location.id}
    className={`p-3 rounded-md transition-colors relative ${
      isSelected
        ? 'bg-tracking-light border-l-4 border-tracking-primary'
        : 'bg-gray-50 hover:bg-gray-100'
    }`}
    onClick={() => onSelectLocation(location)}
  >
    <div className="flex justify-between items-start">
      <div className="flex-1">
        <p className="font-medium flex items-center gap-1">
          <MapPin className="h-4 w-4" />
          GPS Location
        </p>
        <div className="flex items-center gap-2 text-sm text-gray-500">
          <Clock className="h-3 w-3" /> {time}
          <Calendar className="h-3 w-3" /> {date}
        </div>

        <div className="grid grid-cols-2 gap-2 text-xs mt-2">
          <div>
            <span className="text-gray-500">Latitude:</span>
{location.latitude.toFixed(6)}
          </div>
          <div>
            <span className="text-gray-500">Longitude:</span>
{location.longitude.toFixed(6)}
                      </div>
                    </div>

                    <Button
                      variant="outline"
                      size="sm"
                      className="text-xs flex items-center gap-1 mt-2 w-full"
                      onClick={(e) => {
                        e.stopPropagation();
                        window.open(mapUrl, '_blank');
                      }}
                    >
                      <ExternalLink className="h-3 w-3" /> Open in Google Maps
                    </Button>
                  </div>
                </div>
              </div>
            );
          })}
        </div>
      </div>
    </ScrollArea>
  );
};

export default LocationSidebar;
```