Approach to complete the project

**Step 1: Requirement Analysis and Planning (May 21 - May 22)**

Review the project requirements, use cases, and features.

Break down the project into tasks and subtasks.

Define project milestones and deliverables.

Allocate resources and team roles.

Develop a detailed project plan and timeline.

**Step 2: Research and Technology Selection (May 23 - May 24)**

Research relevant technologies for GPS simulation, web development, and security.

Select appropriate tools and frameworks based on project requirements.

Set up development environments and tools.

**Step 3: System Design and Architecture (May 25 - May 26)**

Design the system architecture, including database schema, backend logic, and frontend components.

Define API specifications and data models.

Create wireframes or mockups for the website interface.

Review and finalize the system design with the team.

**Step 4: Implementation (May 27)**

Begin development of the system components according to the defined architecture and design.

Implement user authentication, interactive map interface, vehicle tracking, toll calculation, payment integration, and other key features.

Conduct regular code reviews and testing to ensure quality and adherence to requirements.

**Step 5: Testing and Quality Assurance (May 28)**

Perform comprehensive testing of the developed features, including unit testing, integration testing, and end-to-end testing.

Identify and fix bugs and issues.

Conduct usability testing to ensure a smooth user experience.

Optimize performance and security.

**Step 6: Documentation and Report Preparation (May 29)**

Document the system architecture, installation instructions, API documentation, and user guide.

Compile the project report, including an executive summary, project overview, methodology, results, conclusions, and recommendations.

Review and finalize the report.

Prepare the submission package, including the project files, documentation, and report.

**Step 7: Submission to Intel Team (May 29)**

Package the project files, documentation, and report according to the submission guidelines provided by the Intel team.

Submit the project to the Intel team before the deadline.

Follow up with any additional requirements or clarifications from the Intel team.

**Additional Considerations:**

Communication: Maintain regular communication within the team to track progress, discuss challenges, and provide support.

Contingency Planning: Identify potential risks and develop contingency plans to mitigate them, such as allocating additional resources or adjusting the project timeline if needed.

Time Management: Monitor progress against the project plan and adjust timelines or priorities as necessary to ensure timely completion.

Feedback and Iteration: Solicit feedback from stakeholders and team members throughout the development process and incorporate any necessary revisions or improvements.

By following this plan and closely managing the project's progress, you can ensure the successful completion and timely submission of the GPS toll-based system simulation project to the Intel team.

New Started……………………………………………………………………………………

**Project Plan (May 21 - May 28)**

**May 21: Requirement Analysis and Planning**

* **Define Objectives**: Clearly outline the project goals and desired outcomes.
* **Break Down Tasks**: List all tasks and subtasks required for the project.
* **Timeline**: Create a timeline and set milestones for each major task.

**May 22: Research and Technology Setup**

* **Research Technologies**: Look into the necessary technologies and tools.
  + **Python**: For backend processing and logic.
  + **Flask/Django**: For web framework.
  + **Leaflet.js/Google Maps API**: For interactive maps.
  + **SQLite/MySQL**: For the database.
* **Setup Development Environment**: Install and configure the development environment.

**May 23-24: Design System Architecture**

* **System Architecture**: Design the architecture, including database schema and APIs.
* **Flowchart**: Create a detailed flowchart of the system.

**May 25-26: Implementation**

* **Database Setup**: Create the database schema.
* **Backend Development**: Implement backend logic, including GPS data processing, toll calculation, and user management.
* **Frontend Development**: Develop the web interface for user interaction.
* **Integration**: Integrate frontend and backend components.

**May 27: Testing and Debugging**

* **Unit Testing**: Test individual components for functionality.
* **Integration Testing**: Ensure all components work together seamlessly.
* **User Testing**: Simulate user scenarios to find and fix bugs.

**May 28: Finalization and Documentation**

* **Optimization**: Optimize code for performance.
* **Documentation**: Document the code, create user guides, and compile the final report.
* **Backup**: Ensure all files and data are backed up securely.

**May 29: Submission**

* **Review**: Final review of the project and report.
* **Submit**: Submit the project to the Intel team.

**Detailed Flowchart for GPS-Based Toll Calculation System**

Plaintext

+------------------------------------------------------------+

| Start GPS Toll System |

+------------------------------------------------------------+

|

v

+------------------------------------------------------------+

| Initialize System |

| - Setup database and server |

| - Configure GPS device simulation |

| - Load toll zone boundaries and rates |

+------------------------------------------------------------+

|

v

+------------------------------------------------------------+

| Equip Vehicles with GPS |

| - Simulate GPS devices in vehicles |

| - Ensure devices send periodic location data |

+------------------------------------------------------------+

|

v

+------------------------------------------------------------+

| Monitor Vehicle Locations |

| - Receive real-time GPS data from vehicles |

| - Track vehicle movements within toll zones |

+------------------------------------------------------------+

|

v

+------------------------------------------------------------+

| Detect Entry/Exit Points |

| - Compare GPS data with toll zone boundaries |

| - Record entry and exit times and locations |

+------------------------------------------------------------+

|

v

+------------------------------------------------------------+

| Calculate Toll Charges |

| - Determine distance traveled within toll zones |

| - Apply toll rates based on vehicle type |

| - Factor in dynamic pricing based on conditions |

+------------------------------------------------------------+

|

v

+------------------------------------------------------------+

| Process Payment |

| - Generate toll charge for the trip |

| - Debit amount from user’s account or payment method |

| - Support multiple payment vendors |

+------------------------------------------------------------+

|

v

+------------------------------------------------------------+

| Record Transaction |

| - Update database with transaction details |

| - Provide receipt to user (email/notification) |

+------------------------------------------------------------+

|

v

+------------------------------------------------------------+

| Monitor and Alerts |

| - Real-time monitoring of traffic and system status |

| - Trigger alerts for emergencies or violations |

+------------------------------------------------------------+

|

v

+------------------------------------------------------------+

| Emergency Contingencies |

| - Detect stationary vehicles or accidents |

| - Notify emergency services if needed |

+------------------------------------------------------------+

|

v

+------------------------------------------------------------+

| App and User Interface |

| - Interactive map showing real-time traffic |

| - User login to view trip history and payments |

| - Query system for min/max toll and time savings |

+------------------------------------------------------------+

|

v

+------------------------------------------------------------+

| Reporting and Analytics |

| - Generate reports on toll revenue and traffic |

| - Provide analytics for system optimization |

+------------------------------------------------------------+

|

v

+------------------------------------------------------------+

| End GPS Toll System |

+------------------------------------------------------------+

**Security Features**

1. **Secure Data Transmission**:
   * Use HTTPS to encrypt data transmitted between GPS devices, the server, and user interfaces.
2. **Data Privacy**:
   * Ensure that personal data and vehicle information are securely stored and comply with privacy regulations (e.g., GDPR).
3. **Authentication and Access Control**:
   * Implement strong authentication mechanisms (e.g., OAuth2) and role-based access control.
4. **Regular Audits and Monitoring**:
   * Conduct regular security audits and monitor system logs for suspicious activity.
   * Use intrusion detection systems to identify and mitigate potential threats.
5. **Backup and Recovery**:
   * Implement regular data backups and disaster recovery plans to ensure data integrity and availability.

**Features for the Website**

1. **User Authentication and Authorization**:
   * Secure login for users to manage accounts and view toll history.
   * Admin access for managing the system and viewing comprehensive reports.
2. **Interactive Map Interface**:
   * Display live traffic and toll information on an interactive map.
   * Show real-time vehicle positions and movements.
3. **Real-Time Toll Calculation**:
   * Calculate and display toll charges as vehicles move through toll zones.
   * Provide detailed billing and transaction history.
4. **Dynamic Pricing**:
   * Implement algorithms to adjust toll rates based on traffic conditions and time of day.
5. **Payment Integration**:
   * Support multiple payment methods, including credit/debit cards, mobile payments, and online banking.
   * Secure payment processing and automatic toll deduction options.
6. **Emergency Management**:
   * Alert system for detecting and managing emergencies or violations.
   * Provide support for stalled vehicles or accidents.
7. **Reporting and Analytics**:
   * Generate detailed reports on toll revenue, traffic patterns, and system performance.
   * Provide analytics to help optimize toll rates and manage traffic flow effectively.
8. **User Support and Documentation**:
   * Provide help documentation, FAQs, and customer support options.
   * Ensure users have access to resources for troubleshooting and inquiries.

By following this detailed plan and implementing the flowchart, you can ensure the timely completion and successful deployment of the GPS-based toll system.

**Detailed Plan for GPS Toll-Based System Simulation Using Python**

**May 21: Requirement Analysis and Planning**

**Define Objectives: Clearly Outline Project Goals and Desired Outcomes**

* **Project Goals**:
  + Develop a GPS-based toll system simulation for a 100km x 100km area.
  + Calculate tolls based on vehicle type, distance traveled, and dynamic pricing.
  + Ensure real-time tracking and toll calculation.
  + Provide a user-friendly web interface for vehicle tracking, toll history, and payments.
  + Implement emergency contingencies and multiple payment vendor support.
  + Ensure data security and system reliability.
* **Desired Outcomes**:
  + A functional prototype demonstrating the toll calculation process.
  + A comprehensive report detailing the system’s design, implementation, and performance.
  + A modular and scalable codebase.
  + A secure and user-friendly web application.

**Break Down Tasks: List All Tasks and Subtasks Required for the Project**

1. **Requirement Analysis**
   * Research technologies and tools.
   * Define system requirements and specifications.
   * Identify stakeholders and their needs.
2. **System Design**
   * Design system architecture.
   * Create database schema.
   * Design APIs for GPS data handling and toll calculation.
3. **Setup Development Environment**
   * Install Python and required libraries (Flask/Django, SQLite/MySQL, etc.).
   * Set up a version control system (e.g., Git).
4. **Backend Development**
   * Implement GPS data processing module.
   * Develop toll calculation algorithms (including dynamic pricing).
   * Create user authentication and authorization module.
   * Implement payment processing integration.
5. **Frontend Development**
   * Design web interface using HTML, CSS, and JavaScript.
   * Integrate interactive maps using Leaflet.js or Google Maps API.
   * Develop user dashboard for trip history, toll charges, and payments.
6. **Integration**
   * Connect backend APIs with the frontend.
   * Ensure seamless data flow between GPS devices, server, and web interface.
7. **Testing**
   * Unit testing for individual components.
   * Integration testing to ensure all parts work together.
   * User testing for real-world scenarios.
8. **Security Implementation**
   * Implement HTTPS for secure data transmission.
   * Set up authentication and access control.
   * Conduct security audits and vulnerability assessments.
9. **Emergency Management**
   * Implement real-time monitoring and alert system.
   * Develop protocols for detecting and responding to emergencies.
10. **Reporting and Analytics**
    * Create modules for generating toll revenue and traffic reports.
    * Implement analytics for optimizing toll rates and traffic flow.
11. **Documentation**
    * Write user manuals and technical documentation.
    * Prepare project report detailing design, implementation, and performance.

**Timeline: Create a Timeline and Set Milestones for Each Major Task**

* **May 21**: Requirement Analysis and Planning
  + Define objectives and desired outcomes.
  + Break down tasks and subtasks.
  + Create detailed timeline and milestones.
* **May 22**: Research and Technology Setup
  + Research necessary technologies and tools.
  + Setup development environment and install required software.
* **May 23**: Design System Architecture
  + Design overall system architecture.
  + Create detailed flowchart.
  + Design database schema and APIs.
* **May 24-25**: Backend Development
  + Implement GPS data processing.
  + Develop toll calculation algorithms.
  + Create user authentication and authorization modules.
  + Integrate payment processing.
* **May 26**: Frontend Development
  + Design and develop the web interface.
  + Integrate interactive maps and user dashboard.
* **May 27**: Integration and Testing
  + Connect backend and frontend.
  + Perform unit and integration testing.
  + Conduct user testing for real-world scenarios.
* **May 28**: Finalization and Documentation
  + Optimize code for performance.
  + Implement security features and emergency management protocols.
  + Generate reports and analytics.
  + Write documentation and prepare project report.
* **May 29**: Submission
  + Final review of the project and report.
  + Submit project to the Intel team.

**Detailed Flowchart for GPS-Based Toll Calculation System**

**+----------------------------------------------------+**

**| Start GPS Toll System |**

**+----------------------------------------------------+**

**|**

**v**

**+----------------------------------------------------+**

**| Initialize System |**

**| - Setup database and server |**

**| - Configure GPS device simulation |**

**| - Load toll zone boundaries and rates |**

**+----------------------------------------------------+**

**|**

**v**

**+----------------------------------------------------+**

**| Equip Vehicles with GPS |**

**| - Simulate GPS devices in vehicles |**

**| - Ensure devices send periodic location data |**

**+----------------------------------------------------+**

**|**

**v**

**+----------------------------------------------------+**

**| Monitor Vehicle Locations |**

**| - Receive real-time GPS data from vehicles |**

**| - Track vehicle movements within toll zones |**

**+----------------------------------------------------+**

**|**

**v**

**+----------------------------------------------------+**

**| Detect Entry/Exit Points |**

**| - Compare GPS data with toll zone boundaries |**

**| - Record entry and exit times and locations |**

**+----------------------------------------------------+**

**|**

**v**

**+----------------------------------------------------+**

**| Calculate Toll Charges |**

**| - Determine distance traveled within toll zones |**

**| - Apply toll rates based on vehicle type |**

**| - Factor in dynamic pricing based on conditions |**

**+----------------------------------------------------+**

**|**

**v**

**+----------------------------------------------------+**

**| Process Payment |**

**| - Generate toll charge for the trip |**

**| - Debit amount from user’s account or payment method|**

**| - Support multiple payment vendors |**

**+----------------------------------------------------+**

**|**

**v**

**+----------------------------------------------------+**

**| Record Transaction |**

**| - Update database with transaction details |**

**| - Provide receipt to user (email/notification) |**

**+----------------------------------------------------+**

**|**

**v**

**+----------------------------------------------------+**

**| Monitor and Alerts |**

**| - Real-time monitoring of traffic and system status|**

**| - Trigger alerts for emergencies or violations |**

**+----------------------------------------------------+**

**|**

**v**

**+----------------------------------------------------+**

**| Emergency Contingencies |**

**| - Detect stationary vehicles or accidents |**

**| - Notify emergency services if needed |**

**+----------------------------------------------------+**

**|**

**v**

**+----------------------------------------------------+**

**| App and User Interface |**

**| - Interactive map showing real-time traffic |**

**| - User login to view trip history and payments |**

**| - Query system for min/max toll and time savings |**

**+----------------------------------------------------+**

**|**

**v**

**+----------------------------------------------------+**

**| Reporting and Analytics |**

**| - Generate reports on toll revenue and traffic |**

**| - Provide analytics for system optimization |**

**+----------------------------------------------------+**

**|**

**v**

**+----------------------------------------------------+**

**| End GPS Toll System |**

**+----------------------------------------------------+**

**Detailed Plan for May 23-24: Design System Architecture**

**May 23: Design System Architecture**

**Objective**: Design the overall system architecture, including the database schema and APIs. This will lay the foundation for how different components of the system interact with each other.

**1. System Architecture**

**Components**:

* **Frontend**: User interface for interacting with the system.
  + HTML, CSS, JavaScript (Leaflet.js for maps)
  + Flask templates
* **Backend**: Business logic and database interaction.
  + Flask web framework
  + SQLAlchemy for ORM
  + Geopy for distance calculations
* **Database**: Stores all necessary data.
  + SQLite (for simplicity in development)
* **APIs**: Interfaces for frontend-backend communication.
  + RESTful APIs using Flask

**Interactions**:

1. **User** interacts with the **frontend** via a web browser.
2. **Frontend** sends requests to the **backend** using APIs.
3. **Backend** processes the requests, interacts with the **database** if needed, and returns responses to the **frontend**.

**2. Database Schema**

**Entities**:

* **Vehicle**
  + id (Primary Key)
  + license\_plate
  + vehicle\_type (car, truck, motorcycle, etc.)
  + balance (for toll payments)
* **TollTransaction**
  + id (Primary Key)
  + vehicle\_id (Foreign Key)
  + entry\_point
  + exit\_point
  + distance
  + toll\_amount
  + timestamp
* **TollPoint**
  + id (Primary Key)
  + name
  + latitude
  + longitude

**Schema Diagram**:

plaintext

**+-------------------------+**

**| Start |**

**+-----------+-------------+**

**|**

**v**

**+-------------------------+**

**| User Accesses Website |**

**+-----------+-------------+**

**|**

**v**

**+-------------------------+**

**| User Actions |**

**+-----------+-------------+**

**| - Query Toll from A to B|**

**| - View Dashboard |**

**| - Make a Payment |**

**| - Emergency Contingency |**

**+-----------+-------------+**

**|**

**v**

**+-----------+--------------+--------------+-----------------+**

**| Query Toll from A to B | View Dashboard | Make a Payment | Emergency Contingency |**

**+-----------+--------------+--------------+-----------------+**

**| Input: Start & Destination| Input: Request| Input: Vehicle ID, Amount | Input: Monitor Stationary |**

**| Process: Calculate Route &| Process: Fetch | Process: Deduct Amount | Process: Detect Stationary|**

**| Distance, Fetch Toll Rates| Current Data | Record Transaction | Trigger Alert |**

**| Output: Display Toll & Time| Output: Dashboard| Output: Payment Confirmation | Output: Notify Authorities |**

**+-----------+--------------+--------------+-----------------+**

**|**

**v**

**+-------------------------+**

**| End |**

|  |
| --- |
|  |

**May 24: Create a Detailed Flowchart**

**Objective**: Create a detailed flowchart of the system to visually represent the flow of data and interactions between components.

**Flowchart Components**:

1. **Start**
2. **User Accesses Website**
   * Home Page
3. **User Actions**:
   * Query Toll from Point A to B
   * View Dashboard
   * Make a Payment
4. **Query Toll from Point A to B**:
   * Input: Starting Point, Destination
   * Process: Calculate Route and Distance, Fetch Toll Rates
   * Output: Display Estimated Toll and Time Saved
5. **View Dashboard**:
   * Input: User Requests Dashboard
   * Process: Fetch Current Traffic, Toll Transactions, and Other Stats
   * Output: Display Dashboard with Data Visualization
6. **Make a Payment**:
   * Input: Vehicle ID, Amount
   * Process: Deduct Amount from Vehicle Balance, Record Transaction
   * Output: Display Payment Confirmation
7. **Emergency Contingencies**:
   * Input: Monitor for Stationary Vehicles
   * Process: Detect Stationary Vehicle, Trigger Alert
   * Output: Notify Relevant Authorities
8. **End**

**Detailed Flowchart**:

plaintext

Copy code

+-------------------------+ | Start | +-----------+-------------+ | v +-------------------------+ | User Accesses Website | +-----------+-------------+ | v +-------------------------+ | User Actions | +-----------+-------------+ | - Query Toll from A to B| | - View Dashboard | | - Make a Payment | | - Emergency Contingency | +-----------+-------------+ | v +-----------+--------------+--------------+-----------------+ | Query Toll from A to B | View Dashboard | Make a Payment | Emergency Contingency | +-----------+--------------+--------------+-----------------+ | Input: Start & Destination| Input: Request| Input: Vehicle ID, Amount | Input: Monitor Stationary | | Process: Calculate Route &| Process: Fetch | Process: Deduct Amount | Process: Detect Stationary| | Distance, Fetch Toll Rates| Current Data | Record Transaction | Trigger Alert | | Output: Display Toll & Time| Output: Dashboard| Output: Payment Confirmation | Output: Notify Authorities | +-----------+--------------+--------------+-----------------+ | v +-------------------------+ | End | +-------------------------+

**Detailed Breakdown**

**Query Toll from Point A to B**:

* **Input**: Starting Point, Destination
* **Process**:
  1. Calculate the route using Google Maps API.
  2. Calculate the distance using Geopy.
  3. Determine the toll rate based on vehicle type and dynamic pricing.
* **Output**: Display the estimated toll and time saved.

**View Dashboard**:

* **Input**: User requests the dashboard.
* **Process**:
  1. Fetch the current traffic data.
  2. Retrieve toll transactions from the database.
  3. Generate data visualizations (e.g., graphs, charts).
* **Output**: Display the dashboard with relevant data.

**Make a Payment**:

* **Input**: Vehicle ID, Payment Amount
* **Process**:
  1. Verify the vehicle ID.
  2. Deduct the payment amount from the vehicle's balance.
  3. Record the transaction in the database.
* **Output**: Display payment confirmation.

**Emergency Contingency**:

* **Input**: Monitor for stationary vehicles.
* **Process**:
  1. Detect if a vehicle has been stationary for a predefined period.
  2. Trigger an alert if a stationary vehicle is detected.
  3. Notify the relevant authorities.
* **Output**: Display a notification to the authorities.

**Tools and Technologies**

**Tools**:

* **VS Code**: Code editor
* **Postman**: API testing
* **DBeaver**: Database management

**Technologies**:

* **Frontend**: HTML, CSS, JavaScript (Leaflet.js, Google Maps API)
* **Backend**: Python, Flask
* **Database**: SQLite
* **Libraries**: SQLAlchemy, Flask-Migrate, Geopy

**Conclusion**

By the end of May 24, you should have a clear and detailed system architecture along with a comprehensive flowchart. This will serve as the blueprint for developing and implementing your GPS toll-based system simulation.

**Step 2: Define System Architecture**

In this step, we will:

1. **Design the Database Schema**: Define tables and their relationships to manage vehicles, toll zones, transactions, and user accounts.
2. **Create a Detailed Flowchart**: Visualize the system's workflow, showing how different components interact.

**1. Design the Database Schema**

**Database Schema**

We will use SQLAlchemy for defining the database models. The following tables will be created:

1. **User**: Stores user information.
2. **Vehicle**: Stores vehicle information.
3. **TollZone**: Stores toll zone information.
4. **Transaction**: Stores toll transactions.
5. **GPSData**: Stores GPS data of vehicles.

**ER Diagram**

Here’s a simple representation of the relationships:

scss

Copy code

User (1) ------> (M) Vehicle (1) ------> (M) GPSData ^ | | v | (M) | Transaction v (M) TollZone

**Define Models**

**app/models.py**

python

Copy code

from datetime import datetime from app import db class User(db.Model): id = db.Column(db.Integer, primary\_key=True) username = db.Column(db.String(64), unique=True, nullable=False) email = db.Column(db.String(120), unique=True, nullable=False) password = db.Column(db.String(128), nullable=False) vehicles = db.relationship('Vehicle', backref='owner', lazy=True) class Vehicle(db.Model): id = db.Column(db.Integer, primary\_key=True) license\_plate = db.Column(db.String(32), unique=True, nullable=False) user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False) gps\_data = db.relationship('GPSData', backref='vehicle', lazy=True) transactions = db.relationship('Transaction', backref='vehicle', lazy=True) class TollZone(db.Model): id = db.Column(db.Integer, primary\_key=True) name = db.Column(db.String(64), nullable=False) latitude = db.Column(db.Float, nullable=False) longitude = db.Column(db.Float, nullable=False) radius = db.Column(db.Float, nullable=False) # Radius in meters class Transaction(db.Model): id = db.Column(db.Integer, primary\_key=True) vehicle\_id = db.Column(db.Integer, db.ForeignKey('vehicle.id'), nullable=False) toll\_zone\_id = db.Column(db.Integer, db.ForeignKey('toll\_zone.id'), nullable=False) timestamp = db.Column(db.DateTime, default=datetime.utcnow) amount = db.Column(db.Float, nullable=False) class GPSData(db.Model): id = db.Column(db.Integer, primary\_key=True) vehicle\_id = db.Column(db.Integer, db.ForeignKey('vehicle.id'), nullable=False) latitude = db.Column(db.Float, nullable=False) longitude = db.Column(db.Float, nullable=False) timestamp = db.Column(db.DateTime, default=datetime.utcnow)

**Initialize Database**

**app/\_\_init\_\_.py**

python

Copy code

from flask import Flask from flask\_sqlalchemy import SQLAlchemy from .config import Config app = Flask(\_\_name\_\_) app.config.from\_object(Config) db = SQLAlchemy(app) from app import routes, models

**config.py**

python

Copy code

class Config: SECRET\_KEY = 'your\_secret\_key\_here' SQLALCHEMY\_DATABASE\_URI = 'sqlite:///site.db' SQLALCHEMY\_TRACK\_MODIFICATIONS = False

**manage.py**

python

Copy code

from app import app, db if \_\_name\_\_ == '\_\_main\_\_': db.create\_all() app.run(debug=True)

**2. Create a Detailed Flowchart**

A flowchart helps visualize how different components of the system interact. Here's a high-level flowchart description:

1. **Vehicle Initialization**: Vehicles are initialized with start locations and destinations.
2. **Movement Simulation**: Vehicles move along predefined routes.
3. **GPS Data Collection**: GPS coordinates are collected and stored.
4. **Toll Zone Detection**: Check if vehicles intersect with toll zones.
5. **Toll Calculation**: Calculate toll charges based on distance traveled or zones passed.
6. **Transaction Recording**: Record transactions in the database.
7. **Payment Processing**: Simulate toll payment deduction from user accounts.
8. **Reporting and Visualization**: Generate reports and visualize vehicle movements and toll collections.

**Flowchart Diagram**

Below is a textual representation of the flo

CODE:  
import folium

from geopy.distance import geodesic

from IPython.display import IFrame

# Define toll zones with GPS coordinates (latitude, longitude)

toll\_zones = [

{'name': 'Zone A', 'coordinates': (28.7041, 77.1025), 'radius': 5.0, 'toll\_rate': 10.0}, # Delhi

{'name': 'Zone B', 'coordinates': (19.0760, 72.8777), 'radius': 10.0, 'toll\_rate': 15.0}, # Mumbai

{'name': 'Zone C', 'coordinates': (12.9716, 77.5946), 'radius': 8.0, 'toll\_rate': 12.0} # Bangalore

]

# Define vehicle locations with GPS coordinates

vehicles = [

{'id': 1, 'coordinates': (28.7041, 77.1025)}, # Inside Zone A

{'id': 2, 'coordinates': (19.0760, 72.8777)}, # Inside Zone B

{'id': 3, 'coordinates': (12.9716, 77.5946)}, # Inside Zone C

{'id': 4, 'coordinates': (28.6139, 77.2090)} # Outside all zones (New Delhi)

]

def calculate\_toll(vehicle\_coords, toll\_zones):

total\_toll = 0

for zone in toll\_zones:

distance = geodesic(vehicle\_coords, zone['coordinates']).km

if distance <= zone['radius']:

total\_toll += zone['toll\_rate']

return total\_toll

# Create a map centered around India

m = folium.Map(location=[20.5937, 78.9629], zoom\_start=5)

# Add toll zones to the map

for zone in toll\_zones:

folium.Circle(

location=zone['coordinates'],

radius=zone['radius'] \* 1000, # Convert km to meters

color='blue',

fill=True,

fill\_opacity=0.5,

popup=f"{zone['name']} (Toll Rate: {zone['toll\_rate']})"

).add\_to(m)

# Simulate toll calculation for each vehicle and add to the map

for vehicle in vehicles:

toll\_amount = calculate\_toll(vehicle['coordinates'], toll\_zones)

folium.Marker(

location=vehicle['coordinates'],

popup=f"Vehicle ID: {vehicle['id']} - Toll Amount: {toll\_amount}",

icon=folium.Icon(color='green' if toll\_amount > 0 else 'red')

).add\_to(m)

# Save the map to an HTML file

html\_file = "toll\_zones\_map.html"

m.save(html\_file)

# Display the map within the Jupyter Notebook

IFrame(html\_file, width=700, height=500)