Comprehensive Forest Patrolling Location Tracking App - Complete Development Prompt

📋 Project Overview

Create a highly accurate, real-time location tracking Flutter application for forest patrolling with offline capabilities, multiple tracking modes, and comprehensive monitoring dashboard.

🎯 Core Requirements

1. Authentication & Navigation

- **Login Screen**: Beautiful gradient design with forest theme, dummy credentials (admin/password123)
- **Home Screen**: Permission management, mode selection access, feature overview cards
- **Mode Selection Screen**: Dedicated screen for choosing patrolling method with detailed settings
- **Patrolling Screen**: Real-time monitoring dashboard with comprehensive status information

2. Location Tracking Accuracy (Critical)

- **100% Accurate Coordinates**: No duplicate, invalid, or "jumping" GPS points
- **Multiple Validation Layers**:
    - GPS accuracy filtering (mode-specific thresholds)
    - Speed validation (reject impossible movements)
    - Distance change detection (minimum movement thresholds)
    - Jump detection (reject sudden location teleportation)
    - Median filtering using 3-5 sample buffer
- **Mode-Specific Optimization**: Different settings for walking, cycling, vehicles, etc.

3. Patrolling Modes

Six optimized tracking modes with specific settings:

Mode	Update Interval	Force Send	Max Distance	Max Speed	GPS Accuracy	Suggested Filtering Rules
Walking	3s	10s	30m	15km/h	10m	Ignore speed >15km/h or sudden single-point jumps >20m; require 3+ points' consensus
Cycling	3s	8s	50m	40km/h	10m	Reject points with >30m offset from median; flag speeds >40km/h as potential error
Vehicle	3s	5s	100m	120km/h	10m	Points with error >20m from previous two ignored unless speed is consistent
Running	3s	7s	40m	25km/h	10m	Exclude points with speed spikes >25km/h or GPS accuracy >10m
Motorcycle	2s	4s	200m	100km/h	20m	Ignore sudden jumps >100m or speed >160km/h; apply median smoothing

4. API Integration

Start Patrolling API:

POST: https://cloudbases.in/forest_patrolling/PatrollingAppTestApi/start_patrolling
Parameters: patrolling_name (format: DD-MM-YY-hour-minute-second) Response:
{"status": true, "data": {"patrolling_id": 3}, "message": "Patrolling started successfully"}

Location Update API:

POST:
https://cloudbases.in/forest_patrolling/PatrollingAppTestApi/update_patrolling_track
Parameters: patrolling_id, latitude, longitude, timestamp, sequence_id Response:
{"status": true, "data": {"patrolling_id": id, "latitude": lat, "longitude": lng}, "message":
"Patrolling track successfully updated"}

5. Offline Capabilities & Data Integrity

```
}
```

- **Always Store Locally: Collect and store ALL coordinates regardless of connectivity**
- **Batch Network Operations: Send data to server in batches to minimize network calls**
- **Unique Sequence ID: Generate unique identifier for each coordinate:**
- **Format: MMDDHHMISS (Month+Date+Hour+Minutes+Seconds)**
- **Example: 072214253045 (July 22, 14:25:30.45)**
- **Alternative: Unix timestamp with milliseconds**
- **Timestamp Integration: Send coordinates with precise timestamp to maintain chronological order**
- **Order Preservation: Server receives coordinates with both sequence ID and timestamp for proper ordering**
- **Data Validation**: Only store and sync verified, accurate coordinates

6. Enhanced Data Collection Strategy

javascript

```
// Coordinate Data Structure
{
  "sequence_id": "072214253045",        // Unique ordering identifier
  "timestamp": "2025-07-22T14:25:30.450Z", // ISO timestamp
  "latitude": 8.123456,
  "longitude": 76.654321,
  "accuracy": 12.5,
  "speed_kmh": 25.3,
  "patrolling_id": "3",
  "mode": "cycling",
  "synced": false                    // Local sync status
}
```

7. Batch Sync Strategy

- **Collection Phase**: Store all coordinates locally immediately
- **Batch Processing**: Send coordinates in batches of 20-50 at a time
- **Ordered Transmission**: Sort by sequence_id + timestamp before sending
- **Retry Logic**: Failed batches retry with exponential backoff
- **Progress Tracking**: Show sync progress in real-time dashboard

8. Background Operation

- **Continuous Tracking**: Works when app backgrounded or screen off
- **WorkManager Integration**: Periodic sync every 15 minutes
- **Battery Optimization**: Efficient GPS polling and data processing
- **Proper Lifecycle Management**: Handle app state changes gracefully

🔧 Technical Implementation

Flutter Dependencies (pubspec.yaml)

yaml

```
dependencies:
  flutter:
    sdk: flutter
  geolocator: ^9.0.21
  permission_handler: ^10.4.3
  http: ^0.13.6
  connectivity_plus: ^4.0.2
  sqflite: ^2.3.0
  shared_preferences: ^2.2.0
  workmanager: ^0.5.1
  cupertino_icons: ^1.0.2
```

Android Permissions (AndroidManifest.xml)

xml

```xml
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />

<uses-permission
android:name="android.permission.ACCESS_BACKGROUND_LOCATION" />

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

<uses-permission android:name="android.permission.WAKE_LOCK" />
```

Project Structure

```
lib/
 ├── main.dart
 ├── screens/
│   ├── login_screen.dart
│   ├── home_screen.dart
│   ├── mode_selection_screen.dart
│  └── patrolling_screen.dart
└── services/
    ├── location_service.dart
    ├── api_service.dart
   └── database_service.dart
```

Enhanced Location Validation Logic

dart

```dart
bool _isValidLocation(Position position) {
 // 1. GPS Accuracy Check
```

```
  if (position.accuracy > modeSettings.accuracyThreshold) return false;

  // 2. Coordinate Range Check
  if (position.latitude.abs() > 90 || position.longitude.abs() > 180) return false;

  // 3. Null Island Check
  if (position.latitude == 0.0 && position.longitude == 0.0) return false;

  // 4. Maximum Distance Check (instead of minimum)
  if (lastPosition != null) {
    double distance = calculateDistance(lastPosition, position);
    if (distance > modeSettings.maxDistance) {
      // Reject if movement too large (impossible jump)
      return false;
    }
    // Accept all movements within max distance (including staying still)
  }

  // 5. Speed Validation
  if (calculatedSpeed > modeSettings.maxSpeed) return false;

  return true; // Accept coordinate
}
```

Batch Sync Implementation

dart

```
class BatchSyncManager {
}

static const int BATCH_SIZE = 30;
static const Duration SYNC_INTERVAL = Duration(minutes: 10);

Future<void> syncBatch() async {
  List<Coordinate> pendingCoords = await
database.getPendingCoordinates(BATCH_SIZE);

  // Sort by sequence_id and timestamp
  pendingCoords.sort((a, b) {
    int seqCompare = a.sequenceId.compareTo(b.sequenceId);
    if (seqCompare != 0) return seqCompare;
    return a.timestamp.compareTo(b.timestamp);
  });

  for (Coordinate coord in pendingCoords) {
    bool success = await apiService.sendCoordinate(coord);
    if (success) {
      await database.markSynced(coord.id);
    } else {
      break; // Stop batch if any fails to maintain order
    }

    // Small delay to maintain server order
    await Future.delayed(Duration(milliseconds: 200));
  }
}
```

```
}
```

📊 Real-time Monitoring Dashboard

Statistics Display

- **Total Coordinates**: All captured coordinates (regardless of sync status)
- **Stored Locally**: Count of coordinates in local database
- **Synced to Server**: Successfully transmitted coordinates
- **Pending Sync**: Coordinates waiting for transmission
- **Current Speed**: Real-time speed in km/h
- **Last Coordinate**: Most recent coordinate with timestamp
- **Sync Progress**: Real-time batch sync progress indicator
- **GPS Updates Counter**: Total location captures
- **Connection Status**: Online/Offline indicator
- **Last Update Time**: Timestamp of latest coordinate

Batch Sync Status

- **Sync Queue Size**: Number of coordinates pending upload
- **Last Sync Time**: When last batch was processed
- **Sync Success Rate**: Percentage of successful transmissions
- **Network Status**: Online/Offline with quality indicator

🔒 Updated Data Validation & Quality Assurance

GPS Validation Pipeline (Modified)

1. **Accuracy Check**: Reject coordinates with poor GPS accuracy
2. **Coordinate Validation**: Verify lat/lng within valid ranges
3. **Null Island Check**: Reject (0,0) coordinates
4. **Maximum Distance Validation**: Reject only if movement exceeds realistic maximum

5. **Speed Validation**: Reject physically impossible movements
6. **Buffer Validation**: Median filtering of multiple samples
7. **Always Store Valid**: Store all valid coordinates locally immediately

Efficient Batch Operations

- **Batch Size**: 20-50 coordinates per transmission
- **Smart Timing**: Sync when network quality is good
- **Progressive Sync**: Show real-time progress to user
- **Failure Recovery**: Retry failed batches with backoff

Smart Suggestions System

- **GPS Accuracy Alerts**: Move to open area if accuracy > threshold
- **Connection Issues**: Warnings for server communication problems
- **Offline Mode Status**: Show pending sync count
- **Movement Detection**: Confirm tracking is working properly

Activity Log

- **Timestamped Events**: Real-time activity with exact timestamps
- **Status Messages**: Session start, location updates, server sync, errors
- **Last 5 Activities**: Chronological display with emoji indicators

🎨 UI/UX Design Requirements

Visual Design

- **Forest Theme**: Green gradient backgrounds, nature-inspired colors
- **Modern Cards**: Elevated cards with rounded corners and shadows
- **Responsive Layout**: Adaptive design for different screen sizes
- **Intuitive Icons**: Clear, recognizable icons for all functions

Mode Selection Interface

- **Grid Layout**: 2x3 grid of mode cards with animations

- **Visual Feedback**: Selected mode highlighting and badges
- **Mode Details**: Speed ranges, update frequencies, accuracy info
- **Smart Validation**: Only enable start when mode selected

Real-time Updates

- **Live Counters**: Update statistics every second
- **Progress Indicators**: Visual feedback for all operations
- **Color Coding**: Green for success, orange for warnings, red for errors
- **Monospace Fonts**: Technical data like coordinates and timestamps

🔒 Data Validation & Quality Assurance

GPS Validation Pipeline

1. **Accuracy Check**: Reject coordinates with poor GPS accuracy
2. **Coordinate Validation**: Verify lat/lng within valid ranges
3. **Null Island Check**: Reject (0,0) coordinates
4. **Distance Validation**: Ensure minimum movement between points
5. **Speed Validation**: Reject physically impossible movements
6. **Jump Detection**: Prevent sudden location teleportation
7. **Buffer Validation**: Median filtering of multiple samples

Data Integrity

- **Sequence Numbers**: Maintain proper ordering of coordinates
- **Timestamp Validation**: Ensure chronological consistency
- **Duplicate Prevention**: No repeated coordinates within threshold
- **Error Handling**: Graceful handling of GPS timeouts and failures

🌐 Network & Sync Management

Connection Handling

- **Automatic Detection**: Monitor network connectivity changes
- **Priority Sync**: Always sync offline data before new coordinates

- **Batch Processing**: Efficient bulk sync operations
- **Rate Limiting**: Proper delays between API calls to maintain server order

Offline Strategy

- **Local Storage**: SQLite database with proper indexing
- **Data Queuing**: FIFO queue for pending sync operations
- **Conflict Resolution**: Handle timestamp and sequence conflicts
- **Storage Cleanup**: Remove old synced data periodically

📱 Testing & Validation

Accuracy Testing

- **Walking Test**: 100m walk should generate 8-12 coordinates
- **Vehicle Test**: 12km drive should generate 150-240 coordinates
- **Path Accuracy**: Map should follow actual roads/paths, not straight lines
- **No Zigzag Lines**: Proper chronological ordering prevents erratic paths

Console Debugging

```
🎯 Patrolling mode: Vehicle

📏 Distance: 25.3m, Speed: 45.2 km/h (Vehicle mode)

⏰ Force sending due to Vehicle mode interval (5s)

✅ Valid location (Vehicle): 8.123456, 76.654321 (12.5m, 45.2 km/h)

🌐 Location #15 sent to server successfully
```

Performance Metrics

- **Update Frequency**: Verify mode-specific intervals
- **Battery Usage**: Monitor GPS and network efficiency
- **Memory Usage**: Check for leaks in long sessions
- **Data Usage**: Optimize API call frequency

🔮 Future Enhancement Framework

Extensible Mode Selection

- **Starting Point**: GPS coordinates or address selection
- **User Information**: Name, badge number, team assignment
- **Team Size**: Number of patrol members
- **Equipment Checklist**: Required gear verification
- **Weather Conditions**: Environmental data logging
- **Patrol Route**: Predefined path selection

Advanced Features

- **Geofencing**: Automatic alerts for boundary violations
- **Emergency Alerts**: Panic button with location broadcast
- **Photo Logging**: Geotagged incident documentation
- **Voice Notes**: Audio logging with location stamps
- **Route Analytics**: Statistical analysis of patrol patterns

🎯 Success Criteria

Primary Goals

1. **100% Accurate Tracking**: No duplicate or invalid coordinates
2. **Detailed Path Mapping**: Sufficient points for animation (1 per 50-80m)
3. **Reliable Offline Operation**: No data loss during connection issues
4. **Comprehensive Monitoring**: Real-time status and diagnostics
5. **Mode Optimization**: Appropriate settings for different patrol types

Performance Targets

- **Location Accuracy**: Within 5-10 meters for optimal conditions
- **Update Frequency**: 3-8 seconds based on patrol mode
- **Offline Capacity**: Store minimum 1000 coordinates locally
- **Sync Efficiency**: Process offline data within 30 seconds
- **Battery Life**: 8+ hours continuous operation

💡 Implementation Notes

Critical Success Factors

1. **Test on Real Device**: GPS functionality requires physical hardware
2. **Grant All Permissions**: Especially "Allow all the time" for background location
3. **Proper Mode Selection**: Choose appropriate mode for patrol type
4. **Monitor Console Logs**: Detailed debugging information for troubleshooting
5. **Validate Map Output**: Verify path accuracy on backend map visualization

Common Pitfalls to Avoid

- **LocalStorage Usage**: Not supported in Flutter artifacts
- **Duplicate Coordinates**: Implement proper distance thresholds
- **Wrong Ordering**: Maintain sequence numbers and timestamps
- **Poor GPS Handling**: Implement timeout and fallback mechanisms
- **Background Limitations**: Proper service configuration required