B.M.S. COLLEGE OF ENGINEERING BENGALURU
Autonomous Institute, Affiliated to VTU

OOMD Mini Project Report

Moblie Travel Data Collection & Analysis App

Submitted in partial fulfillment for the award of degree of

Bachelor of Engineering
in
Computer Science and Engineering

Submitted by:

Mukhesh DN (1BM23CS200)
Muppidi Sai Adithya (1BM23CS201)
Namita Devabasappanavar Anilkumar (1BM23CS202)
Nishanth Reddy(1BM23CS214)

Department of Computer Science and Engineering
B.M.S. College of Engineering
Bull Temple Road, Basavanagudi, Bangalore 560 019
2025-26

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, **Mukhesh DN (1BM23CS200), Muppidi Sai Adithya(1BM23CS201), Namita Devabasappanavar Anilkumar(1BM23CS202) ,Nishanth Reddy(1BM23CS214)** students of 5th Semester, B.E, Department of Computer Science and Engineering, BMS College of Engineering, Bangalore, hereby declare that, this OOMD Mini Project entitled "**Moblie Travel Data Collection & Analysis App**" has been carried out in Department of CSE, B.M.S. College of Engineering, Bangalore during the academic semester August 2025- December 2025. I also declare that to the best of our knowledge and belief, the OOMD mini Project report is not from part of any other report by any other students.

Signature of the Candidate

Mukhesh DN (1BM23CS200)

Muppidi Sai Adithya (1BM23CS201)

Namita Devabasappanavar Anilkumar (1BM23CS202)

Nishanth Reddy (1BM23CS214)

B.M.S. COLLEGE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the OOMD Mini Project titled "**Mobile Travel Data Collection & Analysis App"** has been carried out by **Mukhesh DN(1BM23CS200), Muppidi Sai Adithya (1BM23CS201) , Namita Devabasappanavar Anilkumar (1BM23CS202) , Nishanth Reddy(1BM23CS214)** during the academic year 2025-2026.

Signature of the Faculty in Charge (**Sunayana S)**

Table of Contents

Chapter 1: Problem Statement

Understanding how people travel is essential for planning efficient transportation systems and sustainable urban development. However, current methods of collecting travel data are outdated and ineffective. Traditional approaches such as manual surveys, travel diaries, and interviews struggle to provide reliable, large-scale, and up-to-date information.

The major problems with existing travel data collection methods include:

- They are slow, costly, and conducted infrequently.
- Data quality is poor due to forgotten trips, misreported timings, and incomplete information.
- Planners and policymakers lack accurate, real-time mobility insights for decision-making.

At the same time, smartphones offer an opportunity to collect continuous and detailed travel data using built-in sensors like GPS, accelerometers, and gyroscopes. However, several challenges limit the effective use of mobile-based travel tracking:

- **High battery consumption** during continuous location tracking discourages long-term user participation.
- **Difficulty interpreting raw data**, such as identifying trip purpose, transport mode, and user behavior without frequent user input.
- **Privacy concerns**, as location data can reveal sensitive personal information such as home, work, and daily habits.
- **Backend challenges**, including storing, cleaning, and processing large volumes of noisy mobility data and ensuring accuracy.
- **Connectivity issues**, where inconsistent networks can disrupt data transmission.

Due to these limitations, there is a strong need for a **reliable and user-friendly Mobile Travel Data Collection and Analysis system** that can:

- Reduce battery usage while capturing accurate travel patterns
- Protect user privacy through secure and transparent data handling
- Automatically process and interpret mobility data
- Provide high-quality insights for planners and agencies

Without such a solution, cities and transportation authorities continue to face major gaps in understanding real travel behavior, leading to inefficient planning, poor resource allocation, and unsustainable mobility systems

Chapter 2: Software Requirement Specification

1. Introduction

1.1 Purpose of the Document

The purpose of this SRS document is to define the functional and non-functional criteria for creating a mobile application that records trip-related data. By gathering precise, up-to-date trip information from users via an easy and automated procedure, the app seeks to assist NATPAC with transportation planning. Developers, designers, testers, and other project stakeholders can use this paper as a reference.

1.2 Scope of the Document

The goal of this project is to create a mobile travel data collection system that:

- uses smartphone sensors (accelerometer, GPS) to automatically identify journeys.
- records the origin, destination, travel time, mode, distance, and purpose of the journey.
- stores and synchronizes data for analysis with a central NATPAC server.
- gives users a simple way to check and validate their travel information.

The technology will increase the efficiency of transportation planning and research, decrease the workload associated with human surveys, and improve data accuracy.

1.3 Overview

The software will be created for the iOS and Android operating systems. After automatically detecting journeys using GPS and motion sensors, it will ask the user for more information, such as cost and purpose. All data will be saved on the system's secure cloud backend, which NATPAC scientists will be able to access via a web interface. The data gathered will help create a database for research on travel behavior and policy analysis.

2. General Description

2.1 Product Perspective

The product is a mobile-based solution that is connected to a cloud server for managing and storing data. The application will function autonomously, however it will use APIs to communicate with the NATPAC server.
System Elements:

- Mobile application: The user-facing part for data entry and trip recording.
- User and journey data are safely stored on the server backend.
- Web Portal (Admin): Provides authorized access to data that has been gathered for analysis and research.

2.2 Product Functions

Important roles consist of:

1. Consent and User Registration
   -> After creating an account, users consent to data gathering.
2. Trip Detection Automation
   -> The system uses location data to determine the beginning and finish of excursions.
3. Capturing Trip Data
   -> gathers information on origin and destination, start and finish times, method, distance, purpose, cost, and companion count.
4. Data Syncing and Storage
   -> Local data is stored and, when online, synchronized with the cloud.
5. Alerts
   -> Reminders to verify or update trip information are sent by the app.
6. NATPAC Data Access
   -> The dashboard provides administrators with secure access to aggregated travel data.

2.3 User Characteristics

- General Users: People who willingly take part in the gathering of travel-related data.
- Researchers examining combined trip data are NATPAC Scientists/Admins.

2.4 Assumptions and Dependencies

- Before any data is collected, users will give their informed consent.
- For accurate operation, the application needs internet access and GPS.

- Cloud services will be kept up to date and accessible.

3. Functional Requirements

ID	Requirement Description
FR1	The system shall allow users to register and log in using email or mobile number.
FR2	The system shall obtain user consent before data collection begins.
FR3	The system shall automatically detect trip start and end using GPS and accelerometer data.
FR4	The system shall record trip details: trip number, origin (lat/long), destination (lat/long), start and end time, travel distance, mode, purpose, number of companions, and cost.
FR5	The system shall allow users to manually correct or enter missing trip data.
FR6	The system shall synchronize local trip data to a cloud database when connected to the internet.
FR7	The system shall send notifications to remind users to complete trip information.
FR8	The system shall allow NATPAC admins to access, filter, and export collected trip data.
FR9	The system shall allow users to pause or stop data tracking at any time.
FR10	The system shall store all trip data securely and maintain user anonymity.

4. Interface Requirements

4.1 User Interface

- **Mobile App Interface:**
    - Simple and user-friendly layout.
    - Tabs for “My Trips,” “Trip Details,” and “Settings.”
    - Forms to input trip purpose, companions, and cost.
- **Admin Web Interface:**
    - Secure login for NATPAC scientists.
    - Dashboard with search, filter, and export options.

4.2 Hardware Interface

- Mobile phone with GPS, accelerometer, and internet connectivity.
- Cloud server with scalable storage (e.g., AWS, Firebase, or Azure).

4.3 Software Interface

- **APIs:** RESTful APIs for data transmission (JSON format).
- **Operating Systems:** Android (v8.0 and above), iOS (v13 and above).

4.4 Communication Interface

- HTTPS protocol for secure data communication.
- Cloud synchronization when internet access is available.

5. Performance Requirements

- The app should detect trip start/end within **10 seconds** of movement.
- The system should support **at least 1 million users concurrently.**
- Data synchronization delay should not exceed **30 seconds** after connection is restored.
- Average response time for user interactions should be less than **2 seconds.**
- The backend server should maintain **99% uptime**.

6. Design Constraints

- The app must comply with data privacy laws (GDPR, Indian Data Protection Bill).
- Power-efficient GPS tracking to minimize battery usage.
- Device permissions (location access) must be explicitly granted by the user.
- Compatibility with Android and iOS platforms.
- Use of open-source libraries where possible to reduce cost.

7. Non-Functional Requirements

Category	Requirement Description
Security	All user data shall be encrypted both in transit and at rest using HTTPS/TLS.
Reliability	The system shall handle unexpected shutdowns without data loss.
Usability	The app shall have a simple interface requiring minimal user effort.
Maintainability	Modular architecture to allow easy updates and debugging.

Category	Requirement Description
Scalability	The system should scale to support an increasing number of users and data volume.
Portability	Compatible across Android and iOS devices.
Availability	The system shall be accessible 24/7 with regular maintenance windows.

8. Preliminary Schedule and Budget

8.1 Project Schedule (Estimated 6-Month Development Cycle)

Phase	Duration	Activities
Requirement Analysis	2 Weeks	Finalize user and system requirements.
System Design	3 Weeks	Define architecture, UI design, database schema.
Development Phase I	6 Weeks	Implement core modules (registration, GPS tracking, trip detection).
Development Phase II	6 Weeks	Implement trip management, data sync, and admin dashboard.
Testing and Debugging	4 Weeks	Unit testing, integration testing, and performance validation.
Deployment and Training	2 Weeks	Deploy on cloud, train NATPAC staff, finalize documentation.

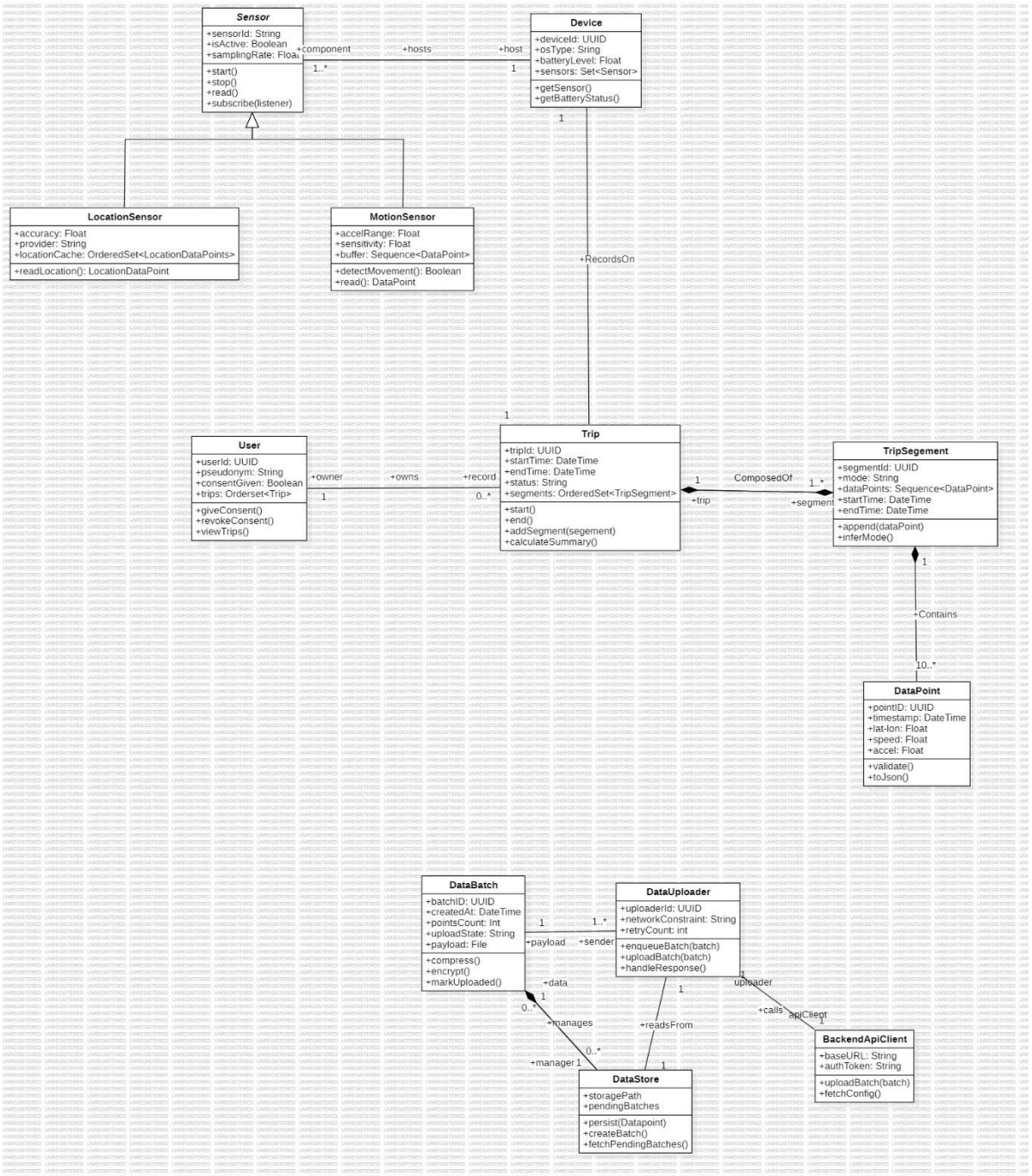
Total Duration: 6 Months

8.2 Estimated Budget

Component	Estimated Cost (INR)
Mobile App Development (Android & iOS)	₹6,00,000
Backend Server & Database Development	₹2,00,000
Cloud Hosting & Storage (1 year)	₹1,00,000
Testing and Quality Assurance	₹1,00,000
Project Management & Documentation	₹50,000
Contingency (10%)	₹1,00,000
Total Estimated Budget	**₹11,50,000**

Chapter 3: Class Modeling

Mobile Travel Data Collection & Analysis App [Class Diagram]



Explaination:

1. Sensor Module

Sensor (Abstract Class)

- Attributes:
    - sensorId, active, samplingRate
- Behaviors:
    - start(), stop(), read()
    - subscribe(listener) → allows other components to listen for sensor events
- **Relationship:**
    - Device *hosts* multiple sensors (1..*)

This is the general template for any hardware sensor used in the device.

LocationSensor (Child of Sensor)

- Attributes:
    - accuracy, provider, locationCache
- Methods:
    - readLocation() → returns a LocationDataPoint

Function: Records GPS location, speed, and route-related data.

MotionSensor (Child of Sensor)

- Attributes:
    - accelRange, sensitivity, buffer
- Methods:
    - detectMovement(), read()

Function: Detects activity patterns such as walking, running, stationary, etc.

2. Device Module

Device

- Attributes:
    - deviceId, osType, batteryLevel, sensors
- Methods:

    - getSensor()
    - getBatteryStatus()
- **Relationships:**
    - Hosts many sensors
    - Records trips on behalf of *User*

The **Device** acts as the central hardware interface connecting sensors and user activities.

3. User & Trips Module

User

- Attributes:
    - userId, username, consentGiven
- Methods:
    - giveConsent()
    - viewTrips()
- Relationships:
    - User *owns* many Trips (1..*)

Users generate trips by moving from one place to another.

Trip

- Attributes:
    - tripId, startTime, endTime, status
    - segments (Ordered Set of TripSegment)
- Methods:
    - start()
    - end()
    - addSegment()
    - calculateSummary()
- Relationships:
    - 1 Trip is composed of multiple Segments

A Trip is a full journey from start to finish for a user.

TripSegment

- Attributes:
    - segmentId, mode (walk, car, bus), dataPoints, startTime, endTime
- Methods:
    - appendDataPoint()
    - inferMode() → may determine transport mode from sensor inputs

A TripSegment is a smaller part of the trip (e.g., walking → bus → walking).

DataPoint

- Attributes:
    - pointId, timestamp, lat, lon, speed, accel
- Methods:
    - validates()
    - toJson()
- Relationship:
    - TripSegment contains multiple DataPoints (0..*)

Represents the raw location/motion data collected at a specific moment.

4. Data Storage & Batch Processing Module

DataBatch

- Attributes:
    - batchId, createdAt, pointsCount, uploadState, payload
- Methods:
    - compress()
    - encrypt()
    - markUploaded()
- Relationship:

- Received from DataStore

Collected data points are grouped into batches for efficient uploading.

DataStore

- Attributes:
  - storagePath, pendingBatches
- Methods:
  - persistDataPoint()
  - createBatch()
  - fetchPendingBatches()
- Relationships:
  - Manages multiple DataBatch
  - DataUploader reads DataBatch from DataStore

This is the device-side storage layer.

DataUploader

- Attributes:
  - uploaderId, networkConstraint, retryCount
- Methods:
  - enqueueBatch()
  - uploadBatch()
  - handleResponse()
- Relationships:
  - Sends DataBatch to BackendApiClient
  - Reads from DataStore

Responsible for network-based uploading of batches.

5. Backend API Communication Module

BackendApiClient

- Attributes:
    - baseURL, authToken
- Methods:
    - uploadBatch()
    - fetchConfig()
- Relationship:
    - Called by DataUploader

Handles communication with backend servers (authentication, config syncing, batch uploads).

6. Summary

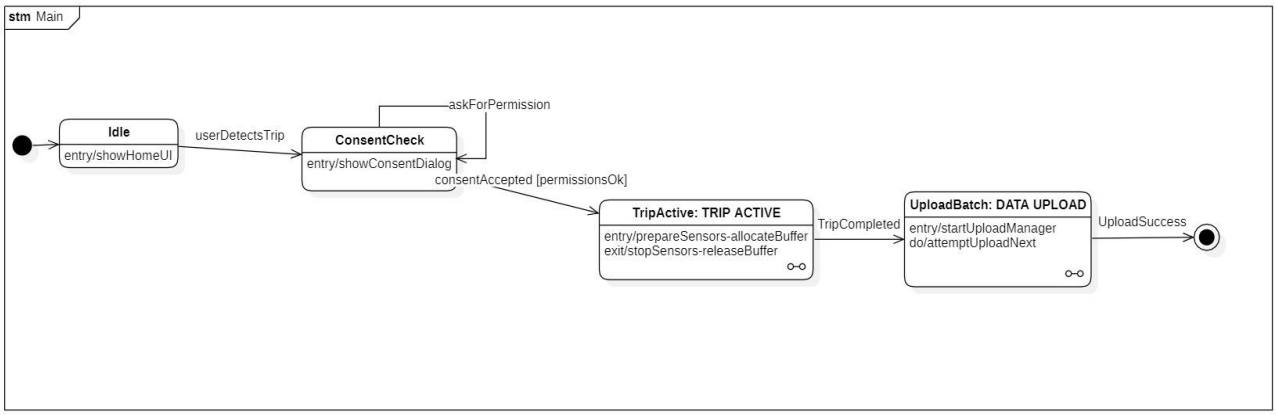
This class diagram models a complete end-to-end data collection framework for a mobile travel tracking system. It covers:

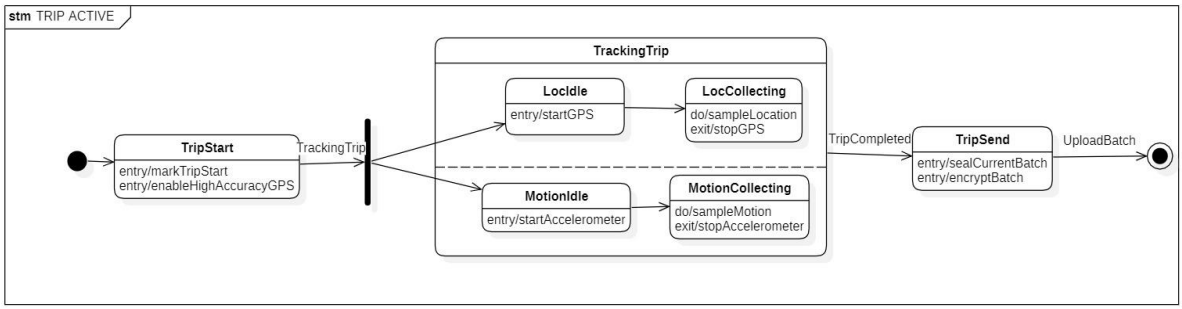
- **Sensors** for data acquisition
- **Trips & Segments** for movement analysis
- **DataPoints** for granular sensor recording
- **DataStore & Batches** for efficient local storage
- **Uploader & API Client** for cloud communication
- **User & Device** context for ownership and hardware interactions

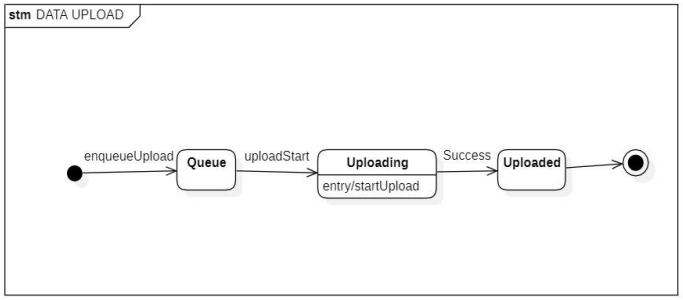
Together, these components enable accurate mobility tracking, segmentation, storage, and secure upload of travel data.

Chapter 4: State Modeling

Mobile Travel Data Collection & Anaylsis App [State Diagram]







Explaination :

1. MAIN STATE MACHINE (stm Main)

This state machine represents the **overall flow of the application** from idle mode → trip tracking → data upload.

State: Idle

- The app is in the default state.
- Shows home UI to the user.
- Waits for the user to start moving or indicate a trip.
- Trigger: **userDetectsTrip**

Transition → **ConsentCheck**

State: ConsentCheck

- Shows a dialog asking the user for:
    - Location permission
    - Motion sensor permission
    - Data collection consent
- Trigger: consentAccepted

If permissions OK → Move to **TripActive**

State: TripActive (Composite State: TRIP ACTIVE)

- Prepares sensors
- Allocates required buffers
- Starts motion & location tracking
- On exit: stops sensors & releases buffers
- Trigger: TripCompleted

Transition → **UploadBatch**

State: UploadBatch (DATA UPLOAD)

- Starts upload manager
- Attempts to upload the next batch of collected data
- Trigger: UploadSuccess

Transition to **End State** (final)

2. TRIP ACTIVE STATE MACHINE (stm TRIP ACTIVE)

This state machine becomes active when the user starts a trip. It manages **GPS tracking**, **motion tracking**, and **trip segmentation**.

State: TripStart

Entry actions:

- Mark the start of a trip
- Enable high-accuracy GPS

Transition → **TrackingTrip**

Composite State: TrackingTrip
This contains **two parallel sub-state machines**:
(A) GPS Tracking Sub-State
LocIdle
- GPS is enabled but idle
- Entry: startGPS

Transition → **LocCollecting**
LocCollecting
- Sampling location data
- Exit: stopGPS

(B) Motion Sensor Sub-State
MotionIdle
- Accelerometer enabled but idle
- Entry: startAccelerometer

Transition → **MotionCollecting**
MotionCollecting
- Sampling motion data (speed, acceleration)
- Exit: stopAccelerometer

Together, these two sub-state machines run **simultaneously**, meaning location and motion data are collected in parallel.

Transition: TripCompleted
When the system detects the user has stopped moving or ended the journey:
Move to:
State: TripSend
Entry:
- Seal the current batch of data
- Encrypt the batch for secure upload

Transition → **UploadBatch (final)**

3. DATA UPLOAD STATE MACHINE (stm DATA UPLOAD)
Handles **queued upload jobs** like batches of travel data.

State: Queue
- Data batches waiting to be uploaded
- Trigger: enqueueUpload

Transition → **UploadStart**

State: UploadStart
- Preparing to upload
- Validates network availability and constraints

Transition → **Uploading**

State: Uploading

Entry action:

- startUpload()

Performs:

- Uploading encrypted batches to the server

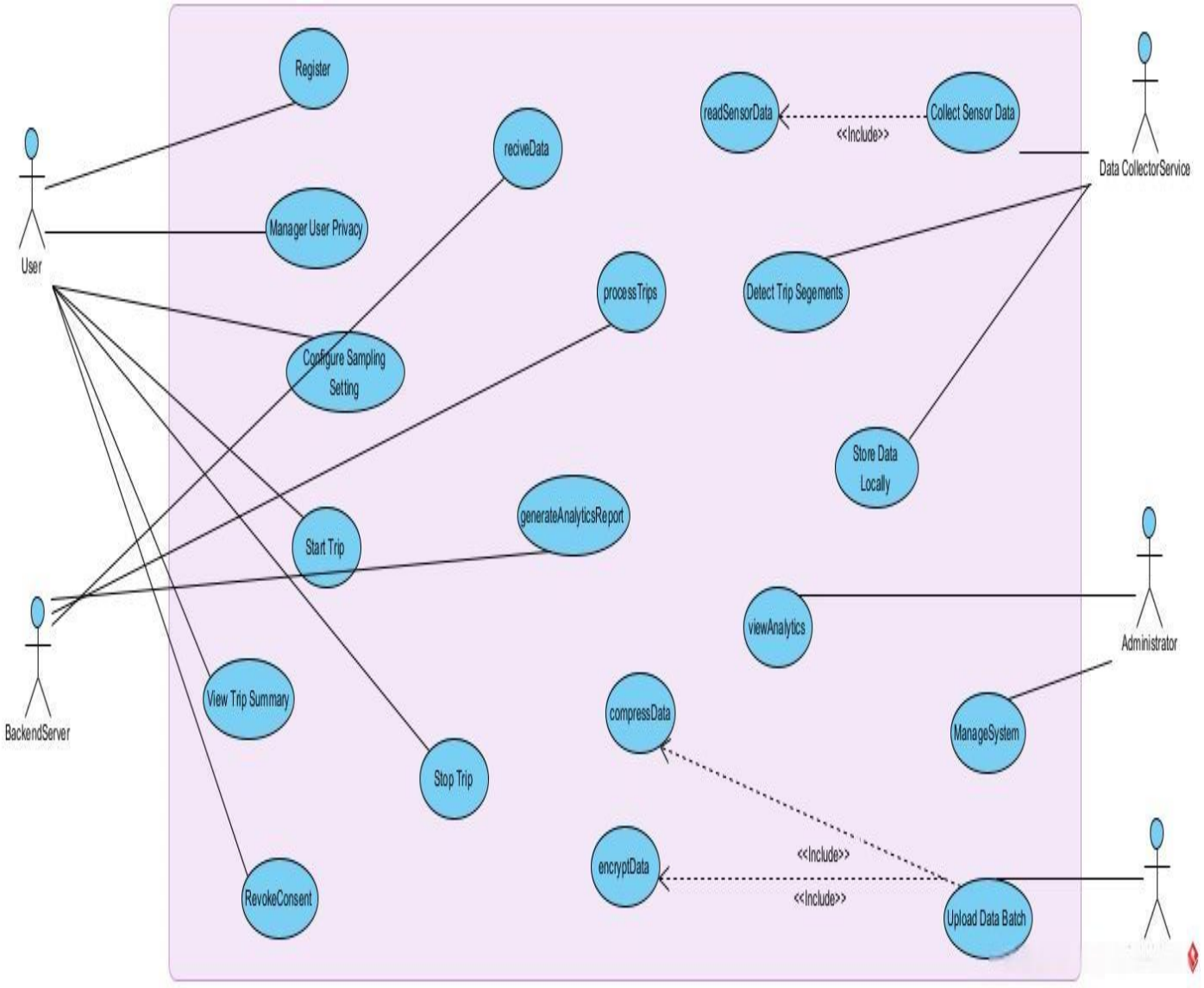
On success → **Uploaded**

State: Uploaded

- Batch is marked as uploaded successfully.
- End state.

Chapter 5: Interaction Modeling

1]Mobile Travel Data Collection & Analyisis App [Use Case diagram]



Explaination :

1. 1. Actors in the System

1. User
A regular app user who:
- Registers and logs into the app
- Starts and stops trips
- Gives consent to capture data
- Configures settings
- Views trip summaries

2. Backend Server
The remote server that:
- Receives uploaded trip data
- Processes and analyzes trips
- Generates reports

3. Data Collector Service (Internal System Component)
A subsystem responsible for:
- Collecting sensor data like GPS & motion
- Detecting trip segments
- Storing data locally

4. Administrator
A system admin who:
- Manages the system
- Views analytics
- Monitors uploaded data

2. Main Use Cases

A. User-related Use Cases

1. **Register**
    - User creates an account to access the system.
2. **Manage User Privacy**
    - User grants or revokes permission for data collection.
    - Includes consent updates.
3. **Configure Sampling Setting**
    - User adjusts GPS/motion sampling rates, tracking frequency, etc.
4. **Start Trip**
    - Begins collecting travel data through sensors.
5. **Stop Trip**
    - Ends the data collection process.
6. **View Trip Summary**
    - Shows user's processed trip information and statistics.
7. **Revoke Consent**
    - User withdraws data-sharing permission; system stops collection.

B. Backend Server Use Cases

1. **receiveData**
    - Server receives uploaded, encrypted, compressed batches of mobility data.
2. **processTrips**
    - Server processes the trip data to extract meaningful segments.
3. **generateAnalyticsReport**
    - Produces aggregated reports for admin or analytics dashboards.

C. Data Collector Service Use Cases

1. **Collect Sensor Data**
    - Responsible for acquiring raw sensor data.
    - Uses:
        - **readSensorData (include)** → Reads accelerometer, gyroscope, GPS.
2. **Detect Trip Segments**
    - Identifies movement patterns (start, stop, pauses, speed, modes).
3. **Store Data Locally**
    - Temporarily stores data on the device before uploading.

D. Administrator Use Cases

1. **viewAnalytics**
    - Admin checks reports, travel statistics, user mobility patterns.
2. **ManageSystem**
    - Handles system settings, dashboards, and operational monitoring.
3. **Upload Data Batch**
    - Involves uploading encrypted/compressed trip data from the device to the server.
    - Includes:
        - **compressData**
        - **encryptData**

3. <Include> Relationships Explained

1. readSensorData → Collect Sensor Data

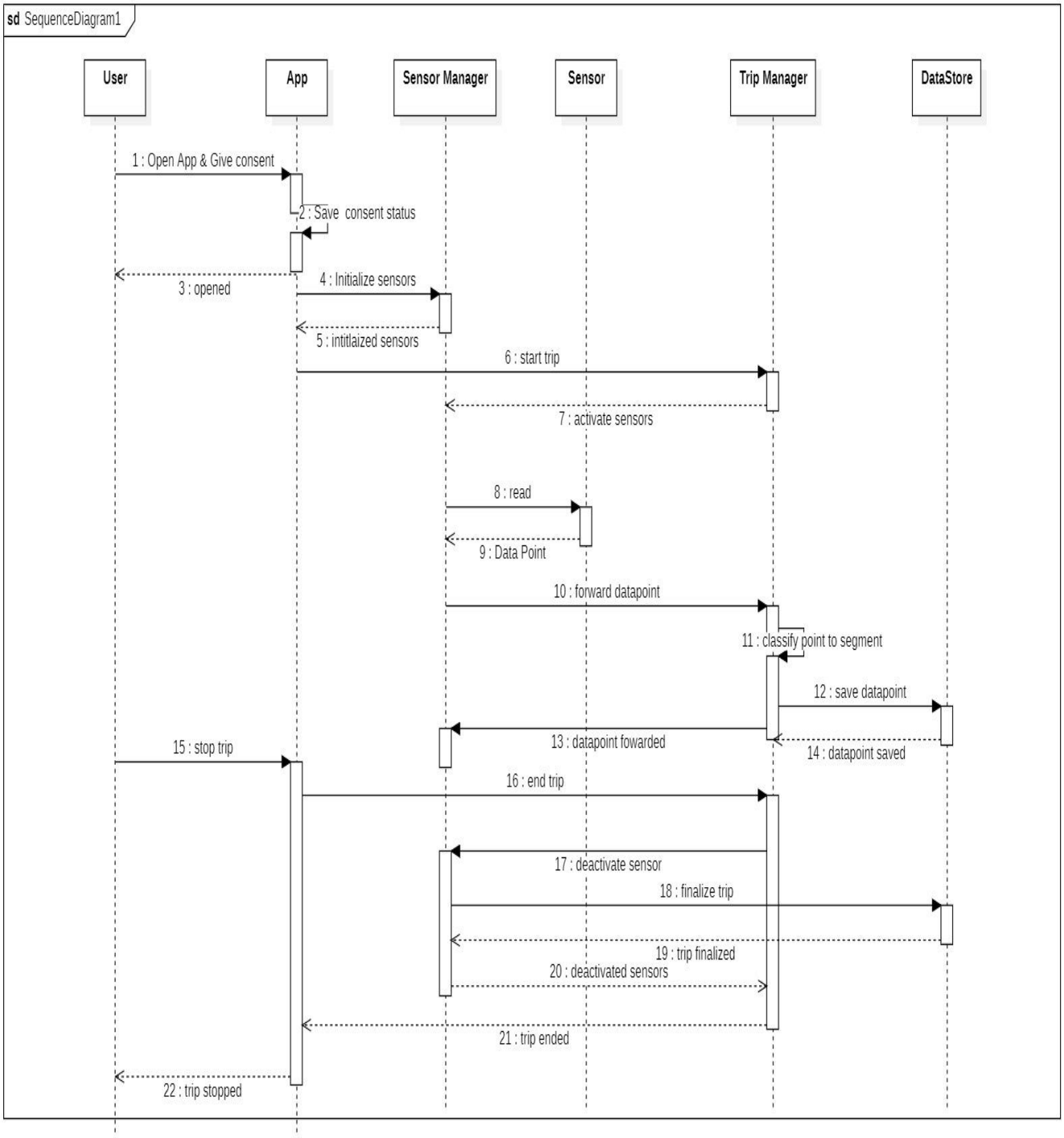
- Sensor data collection depends on reading raw sensor values.

2. compressData & encryptData → Upload Data Batch

- Before uploading:
    - Data must be **compressed** (to reduce size)
    - Then **encrypted** (to secure user data)

These are mandatory subfunctions.

2] Mobile Travel Data Collection & Analysis [Sequence Diagram]



Explaination:

1**. User Interaction and App Initialization**

1. User → App: The user opens the app and gives consent for data collection.
2. App → App (internal): The app saves the user's consent status.
3. App → Sensor Manager: Once the app is opened successfully, it requests the initialization of sensors.
4. Sensor Manager → App: Confirms that sensors are initialized.

2. **Trip Start and Sensor Activation**

5. App → Sensor Manager: The trip is started.
6. Sensor Manager → Sensor: Sensors are activated to begin reading data (GPS, accelerometer, etc.).
7. Sensor → Sensor Manager: The active sensors start generating readings.
8. Sensor Manager → Trip Manager: Each reading is forwarded as a Data Point.

3. **Data Processing and Storage**

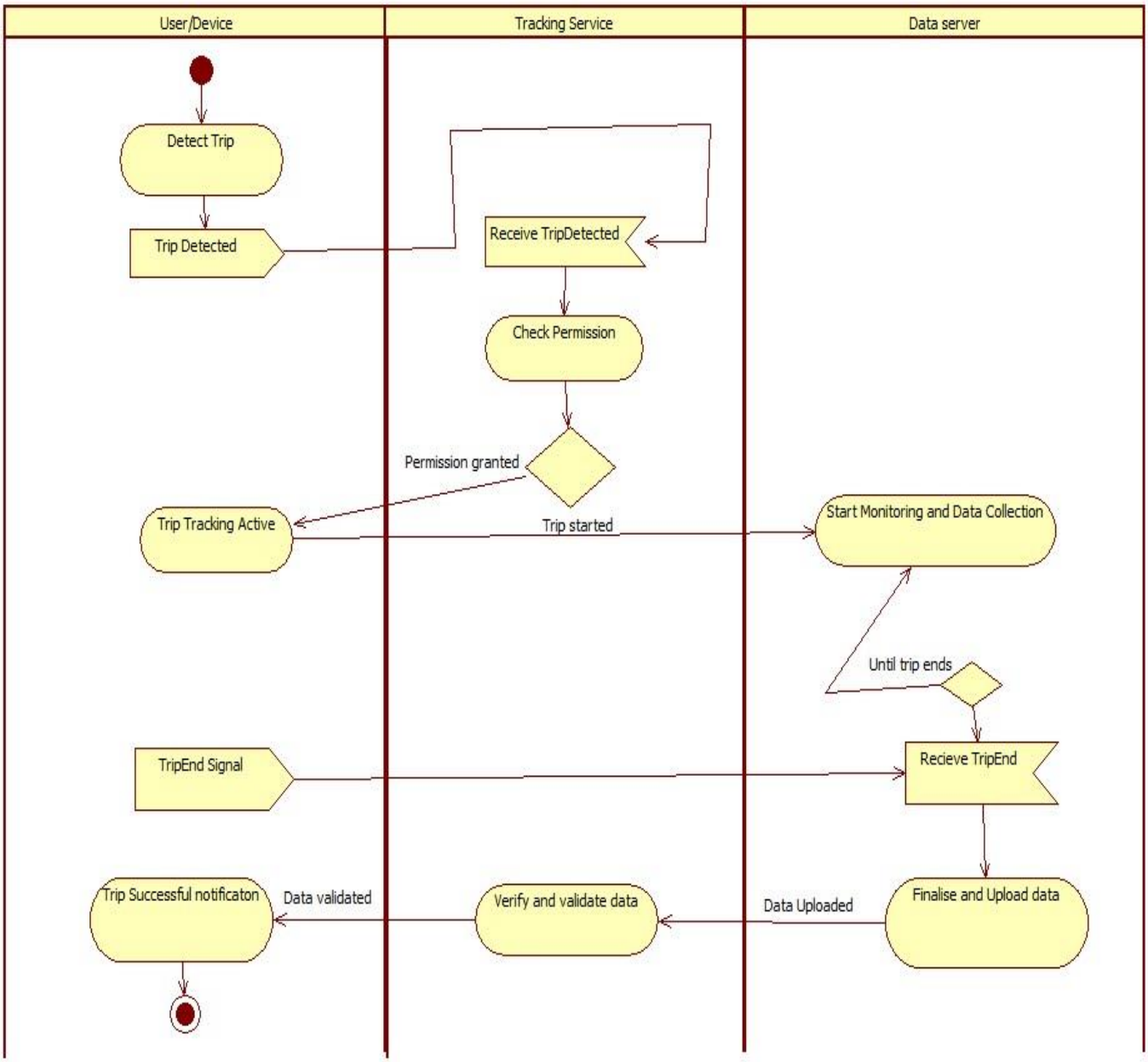
9. Trip Manager → Trip Manager (internal): Classifies the data point into the correct trip segment (walking, vehicle, idle, etc.).
10. Trip Manager → DataStore: The data point is saved locally.
11. DataStore → Trip Manager: Confirms that the datapoint has been stored.
12. Trip Manager → Sensor Manager: Informs that the datapoint is processed and ready for more.

This loop repeats as long as the trip is active.

4. **Trip End and Cleanup**

13. User → App: The user decides to stop the trip.
14. App → Sensor Manager: Requests to end the trip.
15. Sensor Manager → Sensor: Sensors are deactivated.
16. Sensor → Sensor Manager: Confirms sensors are turned off.
17. Sensor Manager → Trip Manager: Notifies that trip has ended.
18. Trip Manager → DataStore: Finalizes the trip, saves summary, and updates records.
19. DataStore → Trip Manager: Confirms trip finalization.
20. Trip Manager → App: Trip ended successfully.
21. App → User: The app notifies the user that the trip has stopped.

3] Mobile Travel Data Collection & Analysis App [Activity Diagram]



Explaination :

1. User/Device Swimlane

A. Detect Trip

- The user's device automatically detects that a trip has started (e.g., based on motion sensors, accelerometer, or user interaction).

B. Trip Detected

- The event "Trip Detected" is sent to the **Tracking Service** to initiate further processing.

C. Trip Tracking Active

- When tracking begins, the device enters the "Trip Tracking Active" state.
- This means the device is now actively recording movement, GPS, and sensor events.

D. TripEnd Signal

- When the device detects the end of the trip (either automatically or by user action), it sends a "TripEnd" signal.

E. Trip Successful Notification

- Once the data is validated by the server and accepted, the user receives a **confirmation** that the trip data was successfully recorded and uploaded.

2. Tracking Service Swimlane

A. Receive TripDetected

- It receives the trip start signal from the device.

B. Check Permission

- Before starting data collection, the service verifies whether:
    - Location permissions are allowed
    - Motion sensor permissions are granted

C. Permission Granted Decision

- If permissions are granted:
    - The system proceeds to start tracking
- If not:
    - It returns to waiting state (represented by the loop arrow)

D. Trip Started

- Once permissions are confirmed, the **Tracking Service** sends a "Start Trip" signal to the **Data Server**.

E. Verify and Validate Data

- After receiving the **TripEnd** event and the upload data from the server:
    - The tracking service validates the structure and integrity of the data
    - Only validated data is pushed to the user device as a confirmation

3. Data Server Swimlane

A. Start Monitoring and Data Collection

- Once it receives the "Trip Started" signal:
    - It begins collecting location and motion data continuously.

B. Loop: Until Trip Ends

- The server keeps monitoring until the trip ends.

- The diamond-shaped decision point checks repeatedly: "Has the trip ended?"

C. Receive TripEnd

- When trip end signal is received from the Tracking Service:
    - Data collection stops.

D. Finalise and Upload Data

- The server processes the collected trip data:
    - Cleans it
    - Segments it
    - Compresses and encrypts it
    - Uploads it to permanent storage

E. Data Uploaded

- The final processed batch is sent back to the Tracking Service for validation.

Chapter 6: UI Design with Screenshots

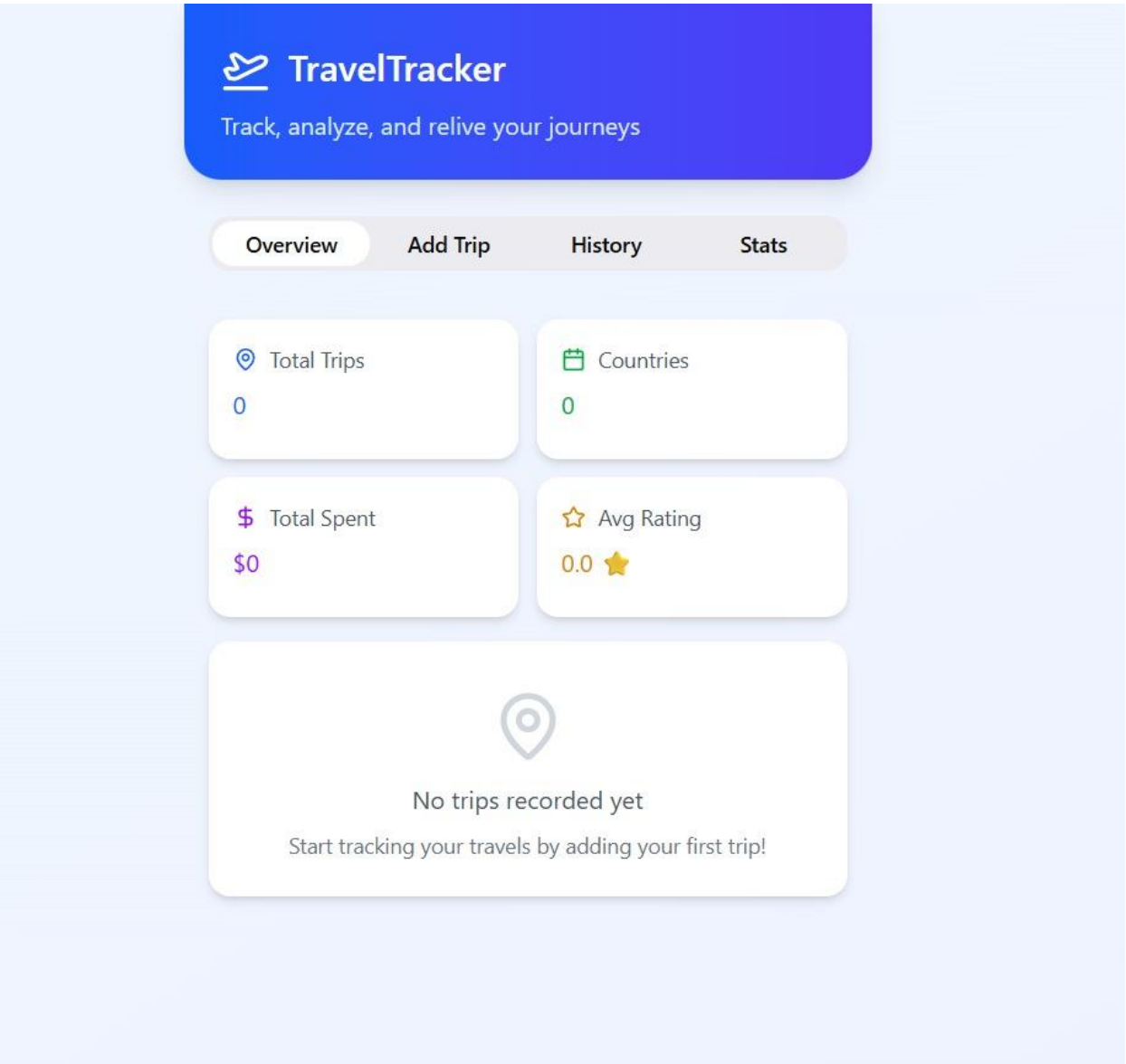


Fig 1.1 [Home Page]

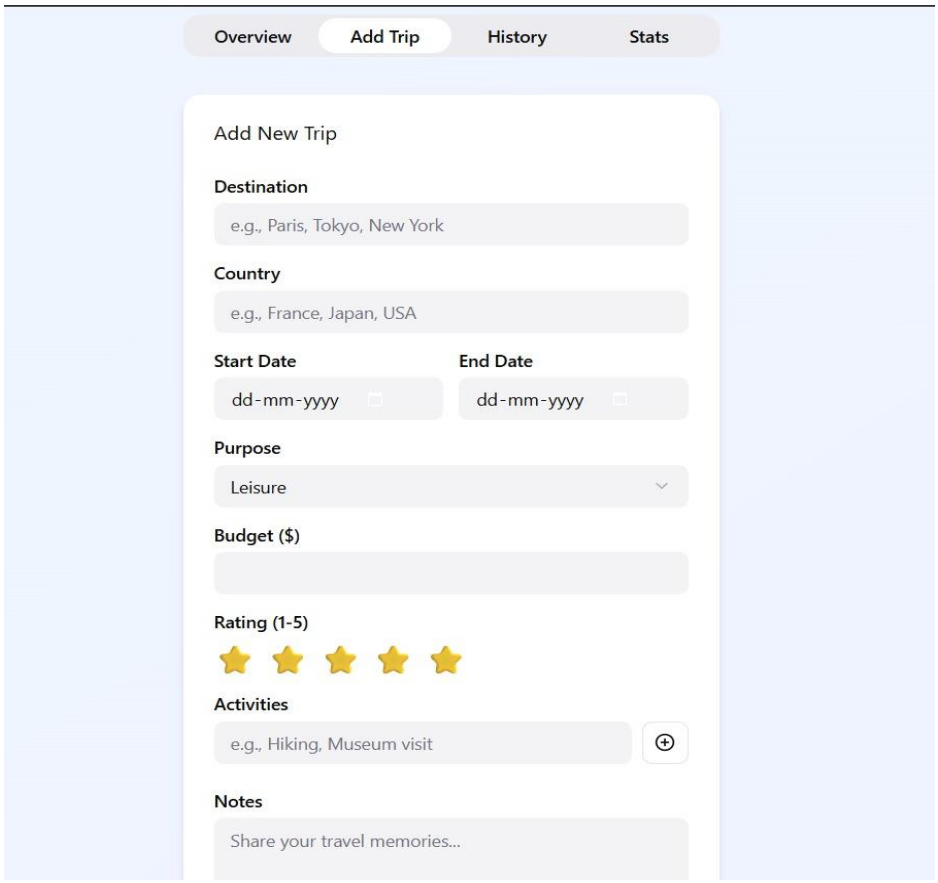


Fig 1.3 [New Trip Additon Page]

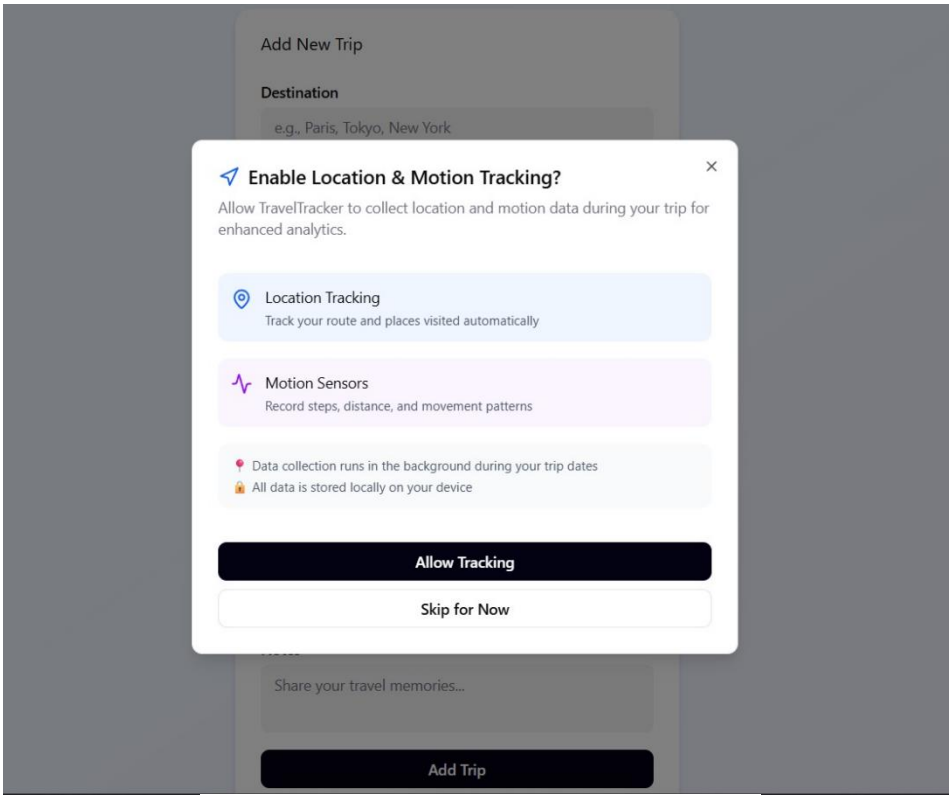


Fig 1.4 [Permission's Approval Page]

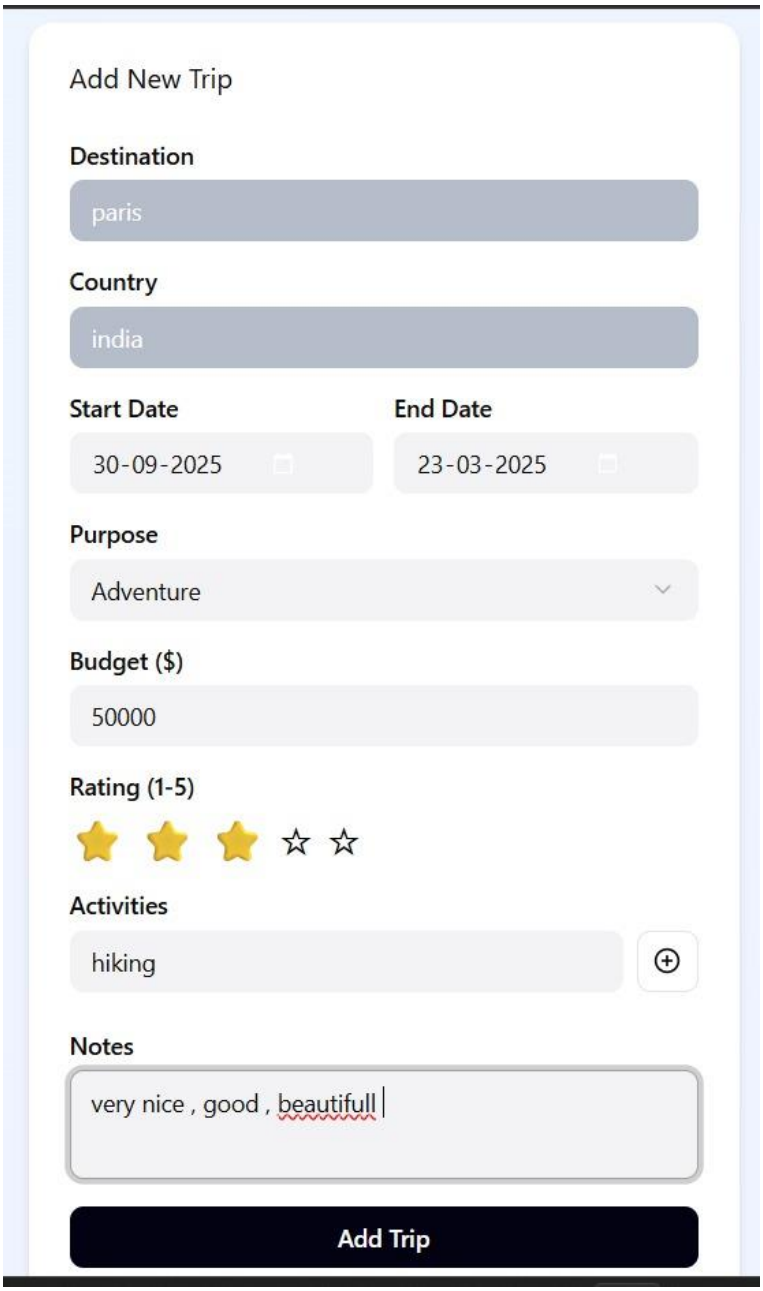


Fig 1.5 [New Trip Addition Page]

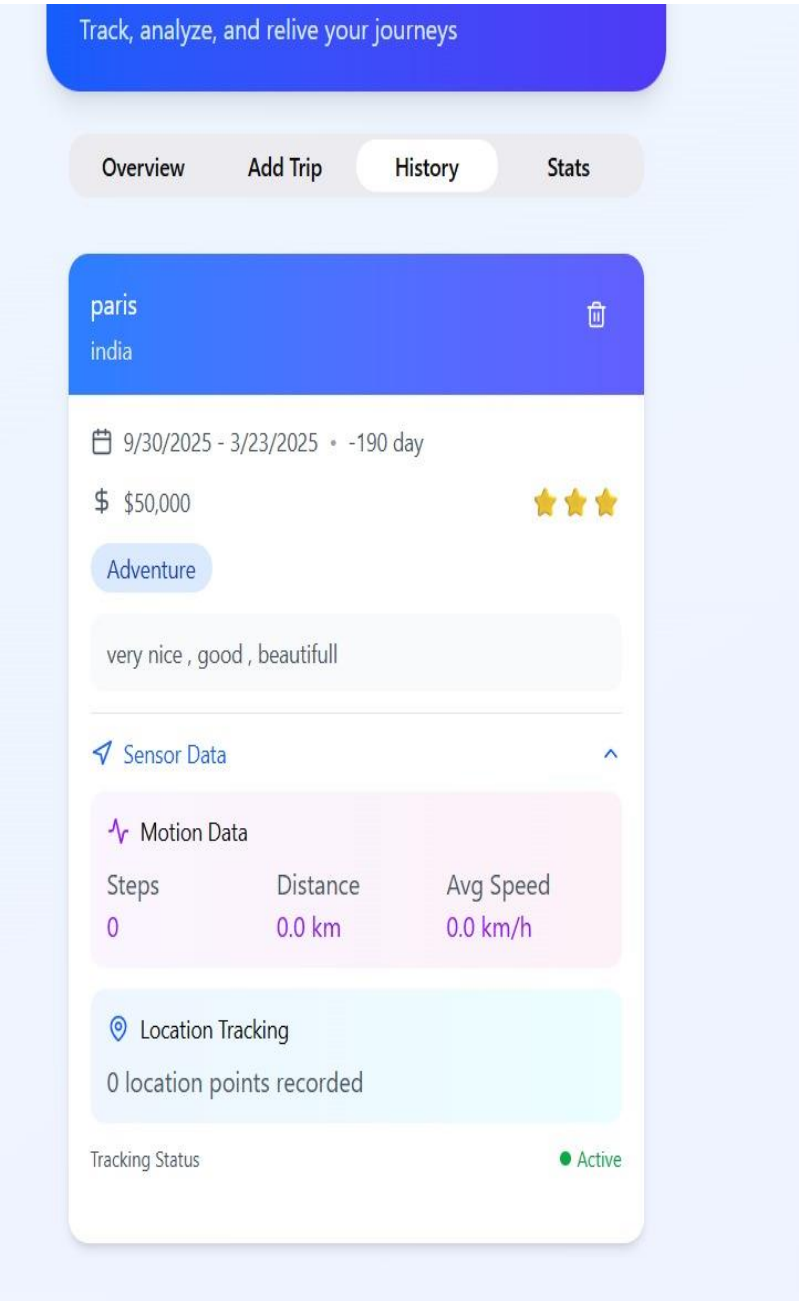


Fig 1.6 [Trip History Analysis Page]

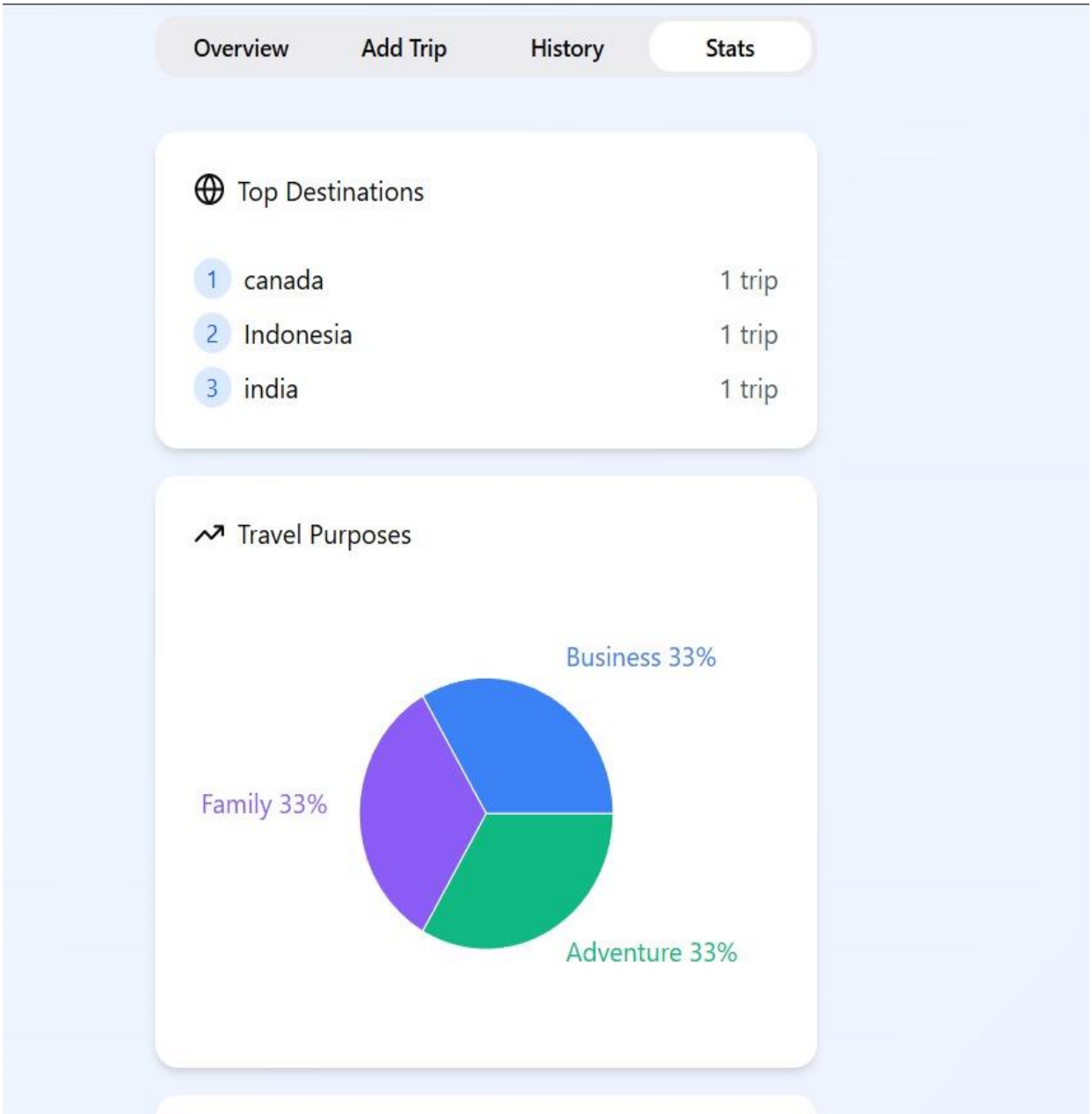


Fig 1.7 [Trip stats Page]

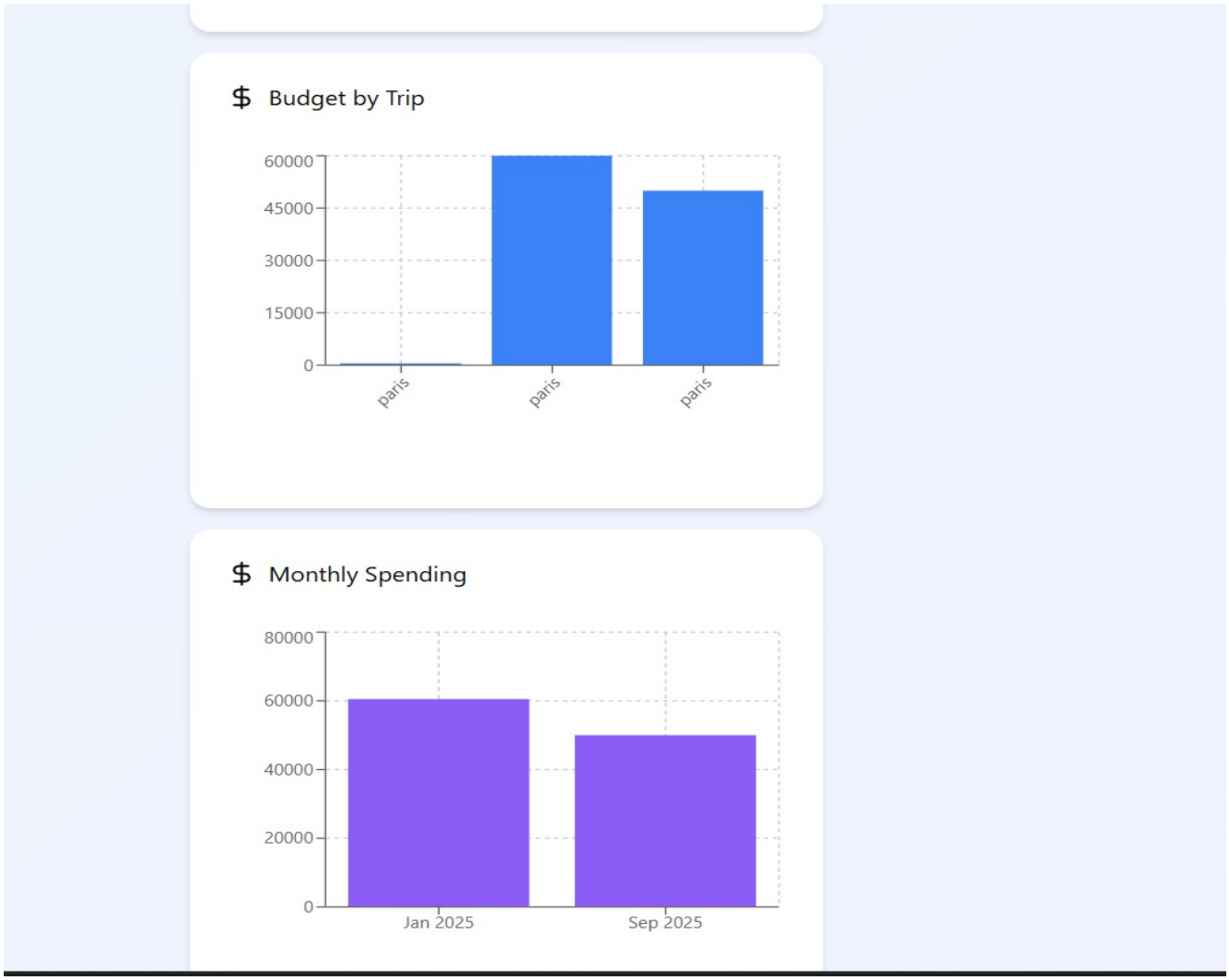


Fig 1.8 [Trip's Captial analysis]

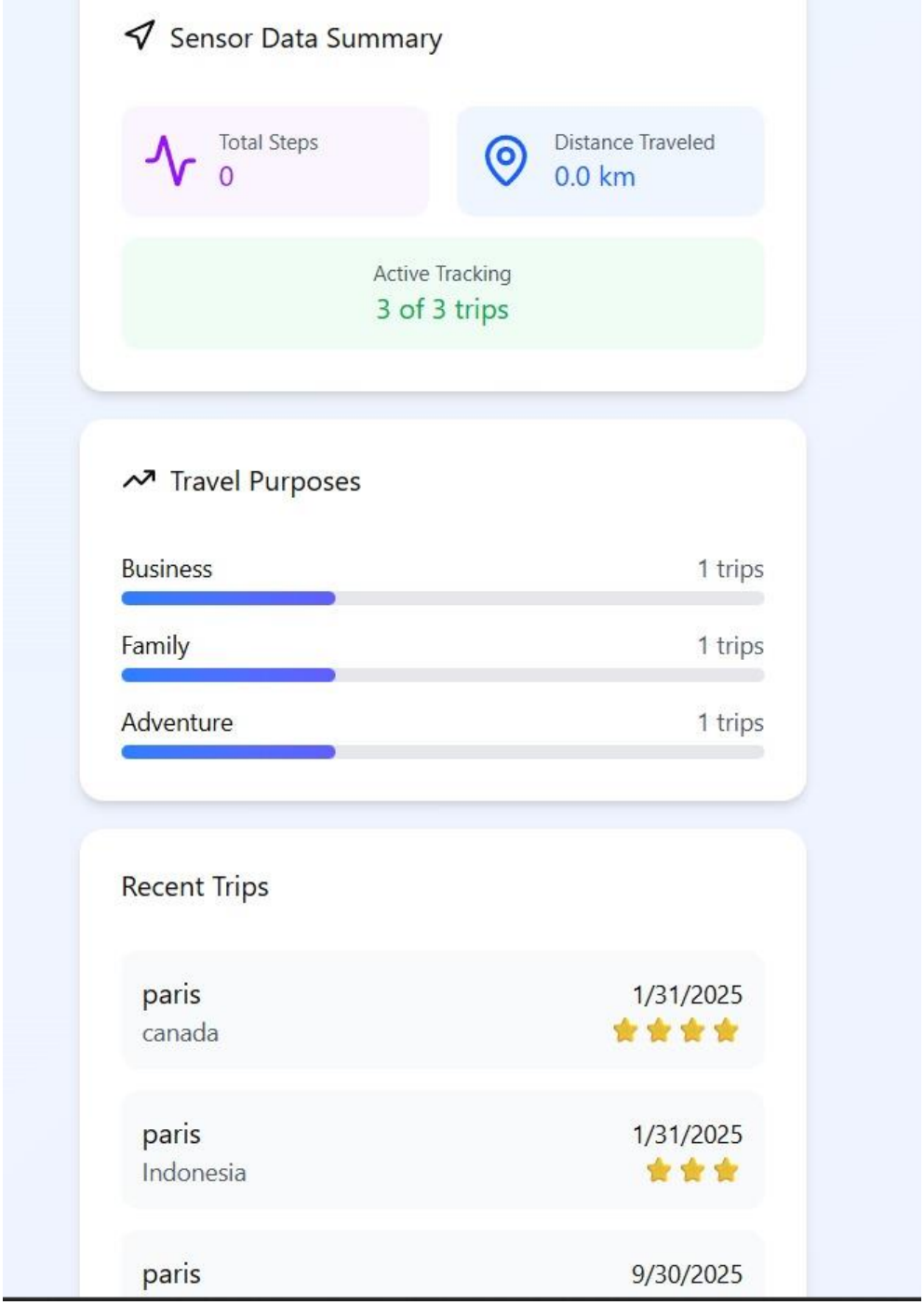


Fig 1.9 [Trip Category analysis page]