**Statement of Requirements (SOR)**

***DiGiSENSE RESTFul API***

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Document Version No** | **Approved Date** | **Approver Name** | **E-Mail Id** | **Remarks** |
| 0.1 |  | Anoop RAJAGOPAL | [RAJAGOPAL.ANOOP@mahindra.com](mailto:RAJAGOPAL.ANOOP@mahindra.com) |  |

This document is to be considered as a whole, the parts of which shall not be separated.

© Mahindra & Mahindra Ltd.  
No duplication permitted without the consent of the issuing department.  
No circulation permitted without the consent of Mahindra & Mahindra.

**Revision History**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ver. No** | **Date** | **Author** | **Description** | **Remarks** |
| 0.1 | 13-10-2016 | Gireesan Namboothiri P. | Initial Document |  |
| 0.2 |  |  |  |  |
| 0.3 |  |  |  |  |
| 0.4 | 02-02-2017 | Gireesan Namboothiri P. | Added Trip status, driver data edit, trip expenses. Prioritized API for first delivery. |  |

**In case of questions about this document and its contents, do not hesitate to contact us**

|  |  |  |
| --- | --- | --- |
| **Name** | **Contact Number** | **E-mail Id** |
| Gireesan Namboothiri P. | 044-22167347  9845487983 | namboothiri.gireesan@mahindra.com |
| Anoop R | 9008151248  044 - 22167904 | [RAJAGOPAL.ANOOP@mahindra.com](mailto:RAJAGOPAL.ANOOP@mahindra.com) |

**Requirement Color Definitions**

|  |  |
| --- | --- |
| **Color** | **Description** |
|  |  |
|  |  |
|  |  |

**Table of Contents**

1. Introduction 5
   1. Purpose of the document 5
   2. Requirements Terminology 5
   3. Confidentiality 6
   4. Coverage of Platform/Vehicles 6
   5. Glossary 6
   6. Terms Definition 6
2. Documents 7
   1. Reference Documents 7
   2. Applicable Documents 8
3. High Level Requirements 9
4. High Level Diagram 9
   1. General System Architecture 9
5. System Requirements 10
   1. Functional Requirement 10
   2. Non-Functional Requirements 51

Annexure: I 52

Annexure: II 52

# Introduction

## Purpose of the document

This document captures the requirements for DiGiSENSE RESTFul API. All the platform functionalities to be extended as RESTFul for flexibility and adaptability.

## Requirements Terminology

The following terminology is used to define the applicability of each requirement in this document.

- The word **Must** in the text means legislative or regulatory requirements (e.g. Health and Safety).

- The word **Shall** in the text means a mandatory requirement.

- The word **Should** in the text means a recommendation or advice on implementing a requirement. Such recommendations or advice are expected to be followed, unless justified reasons are stated for not doing so.

- The word **Will** in the text means an intention or a provision, in connection with a requirement.

**Requirements are noted as follows:**

**Req\_Id Req\_Text**

**Req\_Id** : is the unique requirement identifier of Mid A-IVI EV.

**Req\_Text** : is a description of the requirement.

**Remark:**

- In case of text which doesn’t express a requirement, the word **May** shall be used to denote a permissible practice or action

## Confidentiality

This document is M & M property and contains proprietary and confidential information. This copy is provided on the express condition that its content shall not be reproduced, disclosed to third parties or used, in whole or in part, without the prior written permission of M & M.

## Coverage of Platform/Vehicles

## Glossary

| ***Abbreviation*** | ***Description*** |
| --- | --- |
| *GPS* | *Global Positioning System* |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## Terms Definition

# Documents

## Reference Documents

| **Ref.** | **Reference** | **Version** | **Title / Content** |
| --- | --- | --- | --- |
|  | Mobile Application Manual\_v1 4.pdf | 1.4 | Standardization of API, interfaces, styling and coding standards |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

## Applicable Documents

| **Ref.** | **Reference** | **Version** | **Title / Content** |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# High Level Requirements

In the highest level, the system shall implement RESTFul API with Richardson Maturity model level 2. Level 3 is desirable. All the API requirements are captured with Swagger Tools. [Swagger Editor](http://editor.swagger.io/) to be used to visualize and edit the requirements.

1. Login shall enable Oauth 2.0 with social login support
2. On a successful login, the response will have a [Json Web Token (JWT)](https://jwt.io/).
3. JWT will be used for subsequent requests which provides **authorization** and **authentication**
4. On a failed login, differentiate between login failed (invalid credentials) and user no longer available (user deleted)
5. The API does not differentiate between a specific sector user (eg: FARM/Commercial/MTBD). Instead it should recognize all vehicles mapped under the user name and give responses accordingly.
6. All the requirements and response strings are explicitly captured using Swagger.
7. All the API implementation to be hosted in UAT for testing and validated, security audited APIs to be hosted to the production server
8. Every feature customization for a platform need to be captured using API.
9. On every feature change from the platform, API change has to be initiated, validated with MRV, implemented and tested as per the steps above.
10. Every API to be tested with automation using tools. Eg: Jmeter, POSTMAN
11. Push notifications to be used for **alerts and live route updates**

# High Level Diagram

## General System Architecture

Everything in Green comes under the scope of the API development. The priority of implementation will be specified to make it aligned to the business requirements

Data (JSON): HTTPS

Oauth 2.0

Social Networks

Oauth 2.0

Mahindra

Auth

Token

Authorization

Role based access: JWT

Token

Mobile

Application

Service API (REST)

Token

Token

# System Requirements

## Functional Requirement

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **[Req\_ID]** | **Feature** | **[Requirement\_Text]** | Priority | API |
| 1.01 | Login | Login using Mahindra DiGiSENSE Credentials. Shall return an authentication and authorization token on success | 1.01 | /login  Operation :POST  Request parameters:  {  "username": "admin@example.com",  "password": "secret",  "remember\_me": false  }  Response:  {  jwt: string  serviceId:string    serviceAvailable: [  {  name: string  }  ]  } |
| 1.02 | Login | Login Using Social Login. Shall use Standard Oauth 2.0 handshake. On success, return an authentication and authorization token |  |  |
| 1.03 | Login | Login using Mahindra Oauth Services for DiGiSENSE. . Shall use Standard Oauth 2.0 handshake. On success, return an authentication and authorization token |  |  |
| 1.04 | Login | Social Login should support Google and Facebook integration |  |  |
| 1.05 | Login | Linking of DiGiSense id with Social login credentials to be supported |  |  |
| 1.06 | Login | ForgotPassword: Should use the standard forgot password for DiGiSENSE | 1.02 | /forgotPassword  Operation :POST  Request parameters:  {  username: string  email: string  phoneNumber: integer  }  Response:  {  code: 200  message: success  } |
| 1.07 | Login | resetPassword: Should use the standard reset password for DiGiSENSE | 1.03 | /resetPassword  Operation :POST  Request parameters:  **{**  **username: string**  **email: string**  **phoneNumber: integer**  **newPassword: string**  **otp: string**  **}**  Response:  {  code: 200  message: success  } |
| 1.08 | Login | ChangePassword: Should use the standard change password for DiGiSENSE | 1.04 | /changePassword  Operation :POST  Request parameters:  **{**    **oldPassword: string**  **newPassword: string**  **}**  Response:  {  code: 200  message: success  } |
| 1.07 | Login | ForgotPassword: Should use the standard forgot password for respective login types (DiGiSENSE/Facebook/Google/Mahindra Oauth 2.0) |  |  |
| 2.01 | ServiceAvailable | List all the services available to the current logged in user. Eg: Listing of vehicles, listing of alerts, listing of reports, viewing reports, and track vehicle | 1.05 | /services  Operation: GET  Response:  **[**  string  ] |
| 2.02 | ServiceAvailable | Available service for the user is directly linked to the Role Based Access of the DiGiSENSE platform. Any change in platform through API/WEB interface should reflect the same in service listing |  |  |
| 2.03 | RoleListing | user profileble roles for the current user (differentiate driver and owner) | 1.06 | /userrole  Operation: GET  response:  [“Driver”,”Owner”] |
| 2.04 | RoleCreation | Create a new role (if allowed). Creates the role with a unique name. |  |  |
| 2.05 | RolePermissions | Update permissions for a given role name. This updates the service available |  | /userrole  Operation: POST  Request parameters:  [string]  Response:  [  string,string  ] |
| 2.06 | RoleAssign | Assign roles to a group of users. This include available users (permitted to be listed under current credentials) to be listed and modify (add/delete) role to the user |  |  |
| 3.01 | listVehicles | Get the list of vehicles. All the vehicles under the login with vehiclePlatform, vehicleModel, variant, registration number, current status, last updated timestamp, last known location, priority alert status, current address of the location to be sent.  **If the Platform is MTBD**, additional information to be sent are  Speed  runningHours  driverName | 1.07 | /listVehicles  Operation: GET  Response:  [  {  vehiclePlatform:string  vehicleModel:string  vehicleVariant:string  registrationNumber: string  status: string  lastupdated:string  lastknownlocation:  {  longitude: number  latitude: number  }  priorityAlertStatus:boolean  address: string  speed: string  runningHours: string  driverName: string  }  ] |
| 3.02 | listVehicles | **If the Platform is MTBD**, additional information to be sent are  Speed  runningHours  driverName  This API to return only MTBD vehicles | 1.08 | /listVehicles/mtbd  Operation: GET  Response:  [  {  vehiclePlatform:string  vehicleModel:string  vehicleVariant:string  registrationNumber: string  status: string  lastupdated:string  lastknownlocation: {  longitude: number  latitude: number }  priorityAlertStatus:boolean  address: string  speed: string  runningHours: string  driverName: string  }  ] |
| 4.01 | vehicleSummary | Summary information for the vehicle.  registrationNumber:  vehicleLastUsedOn: | 1.09 | /vehiclesSummary/{vehicleRegNo}  Operation: GET  Request Parameter:  -vehicleRegNo  Response:  [{  registrationNumber: string  vehicleLastUsedOn: string  }] |
| 4.02 | vehicleSummary | **If the Platform is MTBD** information to be sent are  totalUsage:  vehicleRunningHours  vehicleIdleHours | 1.10 | /vehiclesSummary/{vehicleRegNo}  Operation: GET  Request Parameter:  -vehicleRegNo  Response:  {  registrationNumber: string  vehicleLastUsedOn: string  totalUsage: string  vehicleRunningHours: string  vehicleIdleHours: string  } |
| 5.01 | liveVehicleInformation | Get the live vehicle parameters. A client which refresh this API at 1 minute to get latest information. Following information to be returned  registrationNumber  status  lastupdated | 1.11 | /liveVehicleInformation/{vehicleRegNo}  Operation : GET  Request Parameter:  -vehicleRegNo  Response:  {  registrationNumber: string  vehiclePlatform: string  vehicleModel: string  vehicleVariant: string  status: string  lastupdated: string  } |
| 5.02 | liveVehicleInformation | **If the Platform is MTBD** information to be sent are  highEngineTemperature  engineRPM  fuelLevel  vehicleHealth  fuelEfficiencyA  fuelEfficiencyB  averageVehicleSpeed  engineOilPressure  driverName | 1.12 | /liveVehicleInformation/{vehicleRegNo}  Operation : GET  Request Parameter:  -vehicleRegNo  Response:  {  registrationNumber: string  vehiclePlatform: string  vehicleModel: string  vehicleVariant: string  status: string  lastupdated: string  highEngineTemperature: string  engineRPM: integer  fuelLevel: string  vehicleHealth: string  fuelEfficiencyA: string  fuelEfficiencyB: string  averageVehicleSpeed: string  engineOilPressure: string  driverName: string  } |
| 6.01 | Locatedealers | get list of dealers for the current (based on platform and model). The list to be sorted in the distance from vehicle’s current location. Nearest dealer to be the first one in the response. Response should have dealer name, address, phone number, location(latitude, longitude) | 1.13 | /vehicles/{vehicleRegNo}/dealers  Operation: GET  Request Parameter:  -vehicleRegNo  Response:  [  {  dealerId: string  name: string  location: {  longitude: number  latitude: number  }  address:string  contactNumber:integer    }  ] |
| 7.01 | Spares | Get a list of spares for current vehicle (based on platform and model).  Response should include  Sparename  Spareid  Details |  |  |
| 7.02 | Spares | Request for a spare. This should link to the dealer management system to order the spare along with current location of the vehicle.  This should return a request id with nearest dealer’s information where the request has been registered |  |  |
| 7.03 | VehicleAlerts | List of alerts for a given vehicle with  Name, type, location, priority, timestamp. All the alerts specified for the given platform should be enumerated in the above mentioned format | 1.14 | /vehicleAlerts/{vehicleRegNo}  Operation: GET  Request Parameters:   * vehicleRegNo   Response:  [  {    priority: string  alertType:string  dateTime: string  location:{  longitude: number  latitude: number  }    }  ] |
| 8.01 | reportSummary | Get Summary of reports for given platform |  |  |
| 8.02 | ReportSummary/**MTBD** | **If the Platform is MTBD,** report summary should contain  Vehiclemovement  Distance  Current month  Previous month  Total Usage  Current month  Previous month  Idle Time  Current month  Previous month  VehicleHealth  Totalbreakdowns occurred  Total breakdowns closed  Count for each status (Good/Bad/Warming)  Delivery  Planned :  Actual  Utilization percentage current month  Utilization percentage previous month  Number of deliveries current  Number of deliveries previous month  Driver Comparison  Driver Ranking  Distance Covered |  |  |
| 8.03 | Report | Every report information to be enumerated with  Report type (bar,pie, line, table, Label) Name, value series/values. The report should take default week (past 7 days) |  |  |
| 8.04 | Report | MTBD: Vehicle Movement should send the following information  Label : Distance Travelled, value: 1500 KM  Label: Average Speed, value: 45KMPH  Label : Driving Time, value: 250 Hours  Label: Idle Time, value: 15 Hours  Label: Usage Time, value: 225 Hours |  |  |
| 9.01 | GeoFence | Create GeoFence. The API takes 3 or more geographical points as input along with a unique name. Points should be within Indian Political boundaries. None of the points should be in ocean. |  |  |
| 9.02 | GeoFence | Read GeoFence details. This provides points for the named geofence |  |  |
| 9.03 | GeoFence | Update the GeoFence. Add new points. Change existing points. Similar condition checks like 9.01 |  |  |
| 9.04 | GeoFence | Assign GeoFence to one or more vehicles. Parameters are geofence-name, start date, end date, start time,end time, inward/outward and list of vehicles |  |  |
| 9.05 | GeoFence | Delete a named geofence. Cascade to mapping of vehicles to the corresponding geofence |  |  |
| 9.06 | GeoFence | Get Geofences for a given vehicle. Lists currently active geofences (based on the date) |  |  |
| 9.07 | GeoFence | Get vehicles mapped to a given geofence with dates. |  |  |
| 10.01 | RoutePlan | Get all the available plans |  | /routePlan  Operation Method: GET  Response:  [  {  routeId: string  name:string  }  ] |
| 10.02 | RoutePlan | Create a route plan. Routeplan takes options shortest/fastest/custom. In custom, for each point, there will be *associated priority (None/1/2/3/4/5…) , pickup/delivery, location, contact*  Name of the route  Toll Charges and expenses to be entered as “expenses”. |  | /routePlan  Operation Method: POST  Request Parameter:  {  name: string  routeOption: [ String, string]  points: [  {  longitude:number  latitude:number  priority:string  activity:string  address:string  contactNumber:integer  }  ]  }  Response:  {  routeId: string  name: string  routeOption: [ String, string]  points: [  {  longitude:number  latitude:number  priority:string  activity:string  address:string  contactNumber:integer  }  ]  } |
| 10.03 | RoutePlan | Assign Route to one or more vehicles. Parameters are geofence-name, start date, end date, start time,end time and vehicle registration numbers. This creates one trip per vehicle for the given dates. |  | /routePlan/{routeName}/mapVehicle  Operation Method: POST  Request:  {  vehicleRegNo: string  startDate: string  endDate: string  geofenceName: string  status: string  }  Response:  {  id:string  name:string  period: string  routeOption: [ string ]  points: [  routePointId: string  longitude:number  latitude: number  priority: string  activity: string  address: string  contactNumber: integer  ]  assginedVehicles: [ {  vehiclePlatform:string  vehicleModel: string  vehicleRegNo: string  status: string  }  ]  } |
| 10.04 | RoutePlan | View assigned trips for a vehicle. This should include geofences in the route for the date of travel. If it is a completed trip, show geofences active on that dates for the given geographical area of the route. This should include geofences active at that time, but deleted later | 1.15 | /vehicles/{vehicleRegNo}/routes  Operation: GET  Request:  VehicleRegNo  Response:  [  {  routeDetail: {  id: string  name: string  period: string  routeOption:[ string]  points:[ {  routePointId: string  longitude: number  latitude: number  priority: string  activity: string  address: string  contactNumber: integer  }  ]  geofences:[  {  name: string  location:[{ longitude: number  latitude: number  },  { longitude: number latitude: number }  ]  } ]  }  trip: {  tripstatus: string  completed: [ {  activityId: string activity: string plannedLocation: { longitude: number latitude: number }  address: string status: string } ]  toComplete: [ {  activityId: string activity: string plannedLocation: { longitude: number latitude: number } address: string status: string  }  ]  currentLocation: {  longitude: number  latitude: number  }  }  }  ] |
| 10.05 | RoutePlan | Get vehicles (trip details) mapped to the given routeplan |  | /routePlan/{routeName}/vehicles |
| 10.06 | RoutePlan | Update ongoing trip. Similar to Create Route. But applicable with existing points which are not marked with delivery/pick up completed could be either deleted. Or Additional pick up/delivery to be added. This has to create a new entry in the routeplan. |  | /routePlan/{routeName}  Operation Method: PUT  Request:  {  longitude: number  latitude: number  priority:string  activity: string  address: string  contactNumber: integer  }  Response:  {  routeId:string  name: string  routeOption: [ string]  points: [  {  routePointId: string longitude: number  latitude: number  priority:string  activity: string  address: string  contactNumber: integer }    }  ] |
| 10.07 | RoutePlan | Live Trip details. Get the details/status of ongoing trip (started, delivery status, time) | 1.16 | /routePlan/{routeName}  Operation Method: GET  Request:   * route name   Reponse:  {  routeId:string  name: string  routeOption: [ string]  points: [  {  routePointId: string longitude: number  latitude: number  priority:string  activity: string  address: string  contactNumber: integer } |
| 10.08 | RoutePlan | Delete Route. Cascade to mapping of vehicles to the corresponding route for ongoing/upcoming |  | /routePlan/{routeName}:  Operation Method: DELETE  Request:   * route name   Response:  {  code : 200  message: success  } |
| 11.01 | TripManagement | Get active trip for a vehicle (based on the driver login, current date). Returns the status, current waypoints for the route, pickup/delivery details |  | /trips/activetrip/{vehicleRegNo}  Operation Method: GET  Response:  {  tripName: string  tripstatus: string  completed: [  {  activityId: string  activity: string  plannedLocation: { longitude: number  latitude: number  }  address: string  status: string  }  ]  toComplete:  [  {  activityId: string  activity: string  plannedLocation: { longitude: number  latitude: number  }  address: string  status: string  }  ]  currentLocation: {  longitude: number  latitude: number  }  } |
| 11.02 | TripManagement | Status update: start trip. Update the status of trip as **started** | 1.17 | /trip/{tripName}/{vehicleRegNo}/status:  Operation Method: PUT  Request parameter:  -tripName  -vehicleRegNo   * status (started/completed)   Response:  {  tripName: string  tripstatus: string  completed: [  {  activityId: string  activity: string  plannedLocation: { longitude: number  latitude: number  }  address: string  status: string  }  ]  toComplete:  [  {  activityId: string  activity: string  plannedLocation: { longitude: number  latitude: number  }  address: string  status: string  }  ]  currentLocation: {  longitude: number  latitude: number  }  } |
| 11.03 | TripManagement | Status update: complete the trip. Update the status of trip as **completed** | 1.18 | /trip/{tripName}/{vehicleRegNo}/status:  Operation Method: PUT  Request parameter:  -tripName  -vehicleRegNo   * status (started/completed)   Response:  {  tripName: string  tripstatus: string  completed: [  {  activityId: string  activity: string  plannedLocation: { longitude: number  latitude: number  }  address: string  status: string  }  ]  toComplete:  [  {  activityId: string  activity: string  plannedLocation: { longitude: number  latitude: number  }  address: string  status: string  }  ]  currentLocation: {  longitude: number  latitude: number  }  } |
| 11.04 | TripManagement | Status update: completed a delivery. For a given trip for the vehicle, update a delivery/pick-up as **done**.  ActivityID is like 10th point : pickup  Status : done/failed  Done in case of completed and failed if it is not possible in current trip | 1.19 | /trip/{tripName}/{vehicleRegNo}/activity:  Operation Method: PUT  Request parameter:  -tripName  -vehicleRegNo   * status * activityId   Response:  {  tripName: string  tripstatus: string  completed: [  {  activityId: string  activity: string  plannedLocation: { longitude: number  latitude: number  } |
| 11.05 | TripManagement | Live update on the notification in case of trip update from the server.  **Any update on the trip, has to be sent as push notification to the client** | 1.20 | /trip/{tripName}/{vehicleRegNo}  Operation Method: GET  Response:  {  tripName: string  tripstatus: string  completed: [  {  activityId: string  activity: string  plannedLocation: { longitude: number  latitude: number  }  address: string  status: string  }  ]  toComplete:  [  {  activityId: string  activity: string  plannedLocation: { longitude: number  latitude: number  }  address: string  status: string  }  ]  currentLocation: {  longitude: number  latitude: number  }  } |
| 11.06 | TripManagement | Expense Summary. For each trip (on-going/completed), get the expenses (as Label, Value Pairs).  CreatedBy value to be taken from the current login. |  | /tripexpense/{vehicleRegNo}  Operation Method: GET  Request Parameter:  - vehicleRegNo  Response:  [  {  tripName: string  startDate: string  endDate: string  tripStatus: string  expense: [  {  label: string  value: string  }  ]  }  ] |
| 11.07 | TripManagement | Expense Summary. For MTBD, the request should send split values following categories  Fuel consumed  Mileage  Total Kilometers  Self-Expense  Breakdown Count | 1.21 | /tripsummary/{vehicleRegNo}  Operation Method: GET  Request Parameter:  - vehicleRegNo  Response:  [  {  tripName: string  startDate: string  endDate: string  tripStatus: string  expense: [  {  label: string  value: string  }  ]  }  ] |
| 12.01 | BreakDownAssist | Given the vehicle registration number, if the platform permits it, call a breakdown assist. This will cascade to the corresponding Customer Care Centre | 1.22 | /breakDown/{vehicleRegNo}  Operation: PUT  Request parameter:   * vehicleRegNo   Response:  {  longitude: number  latitude: number  timestamp: string Date formated  } |
| 12.02 | BreakDownAssist | Get a list of parts available with dealers. |  |  |
| 12.03 | BreakDownAssist | Get a list of parts available with dealers. For a given spare, get the nearest dealer address, and availability |  |  |
| 12.04 | BreakDownAssist | Order a part. Should accept one value from list of options or “Others”. Additional information to be captured as “description” of service required, nearest landmark/location for arranging the service. |  |  |
| 12.05 | BreakDownAssist | Order a part. Should cascade to dealer management network with selected dealer, location of vehicle, landmark and additional information. Should create a request number for tracking |  |  |
| 13.01 | LocalRepair | Create a local repair entry with date, summary/description, expense amount, repair station details (address, phone) | 1.23 | /localrepair/{vehicleRegNo}  Operation method: PUT  Request parameter:   * vehicleRegNo   Request parameter  {  category: string  detail: string  repairStation: string  repairStationPhoneNumber: string  date: string  amount: string  file: file  }  Response:  {  category: string  detail: string  repairStation: string  repairStationPhoneNumber: string  date: string  amount: string  fileurl: string  } |
| 13.02 | LocalRepair | Get a list of high level local repair types. Eg: Battery, Tyre Puncture, JumpStart… |  |  |
| 13.03 | LocalRepair | Get details of local repair done for the vehicle. “Category/Title: Battery”, Date of repair, amount, repair station details | 1.24 | /localrepair/{vehicleRegNo}  Operation method: GET  Request parameter:   * vehicleRegNo   Reponse:  [  {  category: string  detail: string  repairStation: string  repairStationPhoneNumber: string  date: string  amount: string  fileurl: string  }  ] |
| 14.01 | SelfExpense | Add incidental expenses by the driver. This for an ongoing/completed trip. Inputs will be date, description, amount.  New expenses should accumulate with existing expenses against label/category | 1.25 | /tripexpense/{vehicleRegNo}  Operation Method: PUT  Request:  -vehicleRegNo  {  label:string  value: string  date: string  createdBy: string  }  Response:  [  {  tripName: string  startDate: string  endDate: string  tripStatus: string  expense: [  {  label: string  value: string    }  ]  }  ] |
| 14.02 | SelfExpense | View incidental expenses for ongoing/completed trips | 1.26 | /tripexpense/{vehicleRegNo}  Operation Method: GET  Request Parameter:  - vehicleRegNo  Response:  [  {  tripName: string  startDate: string  endDate: string  tripStatus: string  expense: [  {  label: string  value: string  }  ]  }  ] |
| 15.01 | Reminders | Get all the reminders based on previous configuration for each vehicle |  |  |
| 15.02 | Reminders | For a given vehicle, get all the reminders. Send additional information on fields which are editable for the current user |  |  |
| 15.03 | Reminders | For a given vehicle, update the reminders |  |  |
| 16.01 | LocationHistory | For a given vehicle, get the waypoints along with date and timestamp. Inputs are start date, time and end date time.  In future, the geofences active for that particular travel has to be listed | 1.27 | /locationHistory/{vehicleRegNo}  Operation: GET  Request Parameters:   * vehicleRegNo * startDate * endDate   Response:  [  {  speed:string  address:string  location: {  longitude: number  latitude: number  }  timestamp: string (date-time)  }  ] |
| 17 | Engine management | Get the list of Engine Properties(Engine operating Hours with load,Idle engine hour,Engine temperature,oil pressure) |  | /liveVehicleInformation/{vehicleRegNo}  Operation : GET  Request Parameter:  -vehicleRegNo  Response:  {  registrationNumber: string  vehiclePlatform: string  vehicleModel: string  vehicleVariant: string  status: string  lastupdated: string  highEngineTemperature: string  engineRPM: integer  fuelLevel: string  vehicleHealth: string  fuelEfficiencyA: string  fuelEfficiencyB: string  averageVehicleSpeed: string  engineOilPressure: string  driverName: string  } |
| 18.01 | Driver | Get the list of drivers | 1.28 | /driverProfiles  Operation: GET  Response:  [  {  **id: string**  **firstName: string**  **lastName: string**  **contactNumber: integer**  address: string  **idProof: string**  file: string  email: string  benchmarkFE: string  benchmarkDistance: number  ranking: string  assignedVehicles:  [  string  ]  }  ] |
| 18.02 | profile management | For MTBD, driver profile (add, view, update, edit and delete). | 1.29 | driverProfiles/{driverId}  Operation: PATCH  {  jwt: string  serviceId:string  serviceAvailable: [  {  name: string  }  ]  }  Request Parameters:  - DRIVER ID  [  {  field: string  value: string  type: string  editable: boolean  }  ]  Response:  [  {  id: {  value: string  type: string  editable: boolean  }  firstName: {  value: string  type: string  editable: boolean  }  lastName: {  value: string  type: string  editable: boolean  }  contactNumber: {  value: string  type: string  editable: boolean  }  address:{  value: string  type: string  editable: boolean  }  idProof: {  value: string  type: string  editable: boolean  }  fileUrl: {  value: string  type: string  editable: boolean  }  email: {  value: string  type: string  editable: boolean  }  benchmarkFE: {  value: string  type: string  editable: boolean  }  benchmarkDistance: {  value: string  type: string  editable: boolean  }  ranking: {  value: string  type: string  editable: boolean  }  assignedVehicles: [ string]  }  ]  driverProfiles/{driverId}  Operation: DELETE  Request Parameters:  - DRIVER ID  Response:  {  code: 200  message: success  }  /driverProfiles/{driverId}  Operation: GET  Request Parameters:  - DRIVER ID  Response:  [  {  id: {  value: string  type: string  editable: boolean  }  firstName: {  value: string  type: string  editable: boolean  }  lastName: {  value: string  type: string  editable: boolean  }  contactNumber: {  value: string  type: string  editable: boolean  }  address:{  value: string  type: string  editable: boolean  }  idProof: {  value: string  type: string  editable: boolean  }  fileUrl: {  value: string  type: string  editable: boolean  }  email: {  value: string  type: string  editable: boolean  }  benchmarkFE: {  value: string  type: string  editable: boolean  }  benchmarkDistance: {  value: string  type: string  editable: boolean  }  ranking: {  value: string  type: string  editable: boolean  }  assignedVehicles: [ string]  }  ] |
| 19.01 | profile management | user profile(add,view,update,edit and delete). |  | /profile  Operation :GET  Response:  [  {  field: string  value: string  type: string  editable: boolean  }  ]  Operation : PATCH  Request Parameters:  [  {  field: string  value: string  type: string  editable: boolean  }  ]  Response:  [  {  field: string  value: string  type: string  editable: boolean  }  ] |
| 20.01 | Track vehicle | For a given vehicle, get the list of locations for given date and time. |  |  |
| 21.01 | driver to vehicle mapping | Mapping the driver to vehicle (vehicle list and driver list). Assign mapping based |  |  |
| 22.01 | Alert configuration | Alert configuration to the vehicle (depends upon role  editable). Eg: engine temperature, speed. |  |  |

## Non-Functional Requirements

|  |  |
| --- | --- |
| 1 | All the APIs to undergo security audit |
| 2 | Performance reports on API along with concurrent user lists (10,100,1000, 10000) to be done and made available |
| 3 | Coding standards to be followed based on the documentation |

# Annexure: I

# Annexure: II

END OF DOCUMENT