----------------------VMS----------------------

Introduction : Variable message signs (VMS) are electronic roadside signs used to post traveler information messages to inform drivers of incidents, travel times, detours, special events, and other useful road conditions or travel information.

Scope: For Backend our aim and requirements are VMS can provide information about travel times, delays, and speed limit changes. They can also provide information about public transportation, parking, and route choices.

------------------------------------------------------------------------------------------------------------------------

# Data Base Fields in Database for Vms

1) Id (Unnecessary)

2) UserId (Unnecessary)

3) VmsType Id(Necessary)

4) DeviceId (Unnecessary)

5) DeviceName(Necessary)

6) AssetsModel(Necessary)

7) ListingDate(Necessary)

8) Latitude(Necessary)

9)Longitude(Necessary)

10)Port(Necessary)

11) AssetStatusId(necessary)

12) DateAdded(Necessary)

13)Chainage(Necessary)

14) LocationName(Necessary)

15) CoordinateX(Necessary)

16) CoordinateY(Necessary)

--------------------------------------------------APIs--------------------------------------------------

# Routes:

**1. Create VMS**

**Endpoint: /api/assets/vms**

* **Method**: POST
* **Description**: Creates a new VMS entry in the system.

**2. Update VMS**

**Endpoint: /api/assets/vms**

* **Method**: PUT
* **Description**: Updates an existing VMS entry.

**3. Get All VMS**

**Endpoint: /api/assets/vms**

* **Method**: GET
* **Description**: Retrieves all VMS entries from the system.

**4. Get VMS by ID**

**Endpoint: /api/assets/vms/:id**

* **Method**: GET
* **Description**: Retrieves a specific VMS entry by its ID.

**5. Delete VMS**

**Endpoint: /api/assets/vms/delete**

* **Method**: DELETE
* **Description**: Deletes a VMS entry by its ID.

**6. VMS Types**

**Endpoint: /api/assets/vms/types**

* **Method**: GET
* **Description**: Retrieves the types of VMS available in the system (e.g., static, dynamic)

# Adding Is\_disable feature in PUT Request :

Certainly! In a typical RESTful API design, you can use a PUT request to perform a soft delete by updating the record to mark it as disabled. Here’s how you can implement this directly within your PUT request handler.

**Implementation of PUT Request for Soft Deletion**

Below is a detailed implementation of how you can handle a PUT request to disable (soft delete) a Variable Message Sign (VMS) record in your Go application using the Gin framework.

**Step 1: Modify the VMS Struct**

Ensure your VMS struct has a field for soft deletion, such as IsDisabled:

type Vms struct {

Id int `json:"id" gorm:"primary\_key"`

UserId int `json:"user\_id"`

VmsTypeId int `json:"vms\_type\_id" binding:"required"`

DeviceID string `json:"device\_id" binding:"required"`

DeviceName string `json:"device\_name" binding:"required"`

AssetsModel string `json:"assets\_model" binding:"required"`

ListingDate string `json:"listing\_date" binding:"required"`

Latitude string `json:"latitude" binding:"required"`

Longitude string `json:"longitude" binding:"required"`

Ip string `json:"ip" binding:"required"`

Port string `json:"port"`

AssetStatusId int `json:"asset\_status\_id" binding:"required"`

DateAdded string `json:"date\_added"`

Chainage string `json:"chainage"`

LocationName string `json:"location\_name"`

CoordinateX int `json:"coordinate\_x"`

CoordinateY int `json:"coordinate\_y"`

IsDisabled bool `json:"is\_disabled" gorm:"default:false"` // New field for soft delete

}

**Step 2: Create the PUT Request Handler**

Step 3: Define the Route

router.PUT("/vms/:id/disable", DisableVms)

Step 4: In-Term Of Executing and Running the Application

**i)On Success**:

{

"message": "VMS record has been disabled"

}

**ii)On Not Found**:

{

"error": "VMS not found"

}

iii) On Error:

{

"error": "Failed to disable VMS"

}

**Additional Documentation Fields**

* **Purpose**: The purpose of the Vms struct is to represent a Variable Message Sign system within a broader application, allowing for tracking, management, and operational control of VMS devices in a networked environment.
* **Usage**: This struct can be used in API requests/responses to create, read, update, and delete (CRUD) operations for VMS records. It integrates with a database via GORM for persistent storage.
* **Validation**: Required fields should be validated in API requests to ensure integrity and correctness. The use of binding tags like binding:"required" assists in enforcing this validation.
* **Database Integration**: The struct is designed for use with GORM, a popular ORM for Golang, which allows for easy interaction with SQL databases.
* **Communication**: The Ip and Port fields are critical for establishing network communications with the VMS, allowing for real-time updates and command control.