ABCDE Internship

**Project name : shoppingCart**

**Tech Stack :** 1. Frontend – React  
 2. Backend – Go  
 3. Database – MySQL  
**Backend**:  
 1. User signup - /users  
 2. User login - /users/login  
 3. Create item - /item  
 4. List items - /items  
 5. Add items to cart - /carts  
 6. View Cart - /carts  
 7. Placing an order - /orders  
 8. View the placed order - /orders

**Step-by-Step Approach:**

1. Installing Go (golang)
2. Open cmd and type ‘**go version’** to ensure the installaization
3. Next move to ‘**D:**’ drive and make a folder ‘**mkdir shoppingCart**’
4. Navigate to the project folder ‘**cd shoppingCart’**
5. Install the Required Go packages:   
   **go get github.com/gin-gonic/gin –** for installing gin web framework  
   **go get gorm.io/gorm –** install GORM for ORM/database access  
   **go get github.com/glebarez/sqlite–** Add sqlite driver(so need not install a separate DB server)
6. Creating a User Model:

i)create a file: **mkdir model**

ii)navigating to the folder: **cd model**

iii)creating an empty file: **type nul > model.go** (if exists, it overwrites with nothing)

* Opening the model.go in vs code

package model

import "gorm.io/gorm"

type User struct{

gorm.Model

username string `gorm: "unique"`

password string

token    string

}

1. Connecting to SQLite using GORM:

Just like defining a **DataSource,Entity or application,properties** in Spring Boot.

In **Go + GORM** – create a DB connection in code.

i) creating a folder : **mkdir database**ii) navigating to the folder: **cd database**iii) creating an empty file: **type nul > data.go**

iv) gorm.Open() – opens a connection to **cart.db** file  
v) AutoMigrate() – creates or updates the user table based on our struct

* Opening the data.go in vscode  
    
  package database

import(

"log"

"gorm.io/gorm"

"github.com/glebarez/sqlite"

"shoppingCart/model"

)

var DB \*gorm.DB

func InitDB(){

var e error

DB, e=gorm.Open(sqlite.Open("cart.db"), &gorm.Config{})

if e!=nil{

log.Fatal("Failed to connect to database: ",e)

}

DB.AutoMigrate(&model.User{})

}

1. Creating the **main.go** file:

i)navigate to the main folder: **cd D:\shoppingCart**

ii)create the empty file: **type nul > main.go**

iii)opening the main.go in vscode

package main

import(

    "shoppingCart/database"

    "shoppingCart/model"

    "github.com/gin-gonic/gin"

    "net/http"

)

func main(){

    database.InitDB()

    router := gin.Default()

    router.POST("/users", func(c \*gin.Context){

        var user model.User

        if e := c.ShouldBindJSON(&user); e!=nil{

            c.JSON(http.StatusBadRequest, gin.H{"error": "Invalid"})

            return

        }

        database.DB.Create(&user)

        c.JSON(http.StatusOK, gin.H{

            "msg": "User created",

            "user": user,

        })

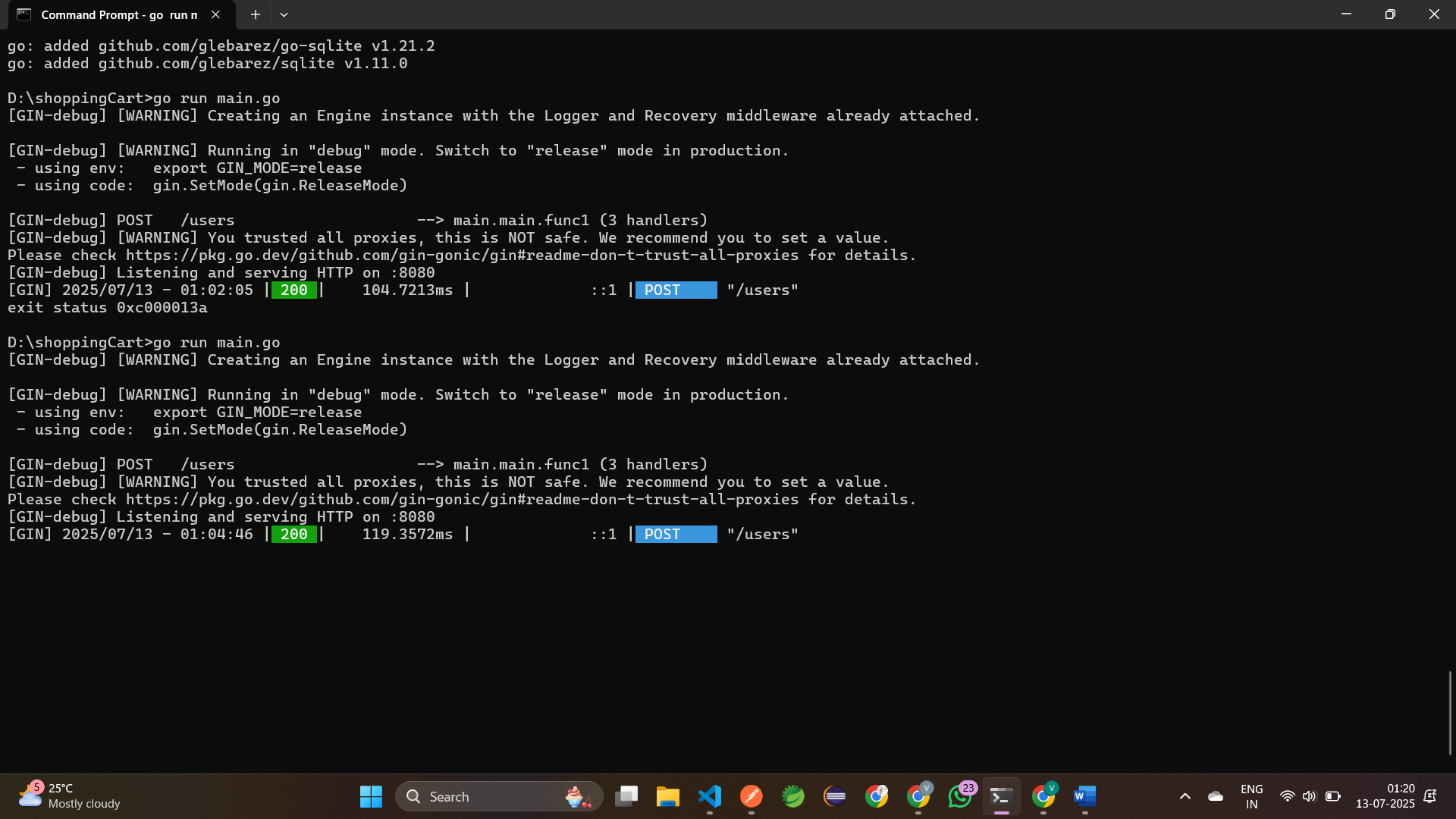
    })

    router.Run(":8080")

}

1. Running the application:

**go run main.go**



10. Testing the POST method in POSTMAN:

Url: [**http://localhost:8080/users**](http://localhost:8080/users)

**ERROR OCCURRED:**

When sending input on the Body in raw form , in ouput the variables declared in **model.go** does’nt display.

INPUT:

{ "username": "vikas",

"password": "1234"

}

OUTPUT:

{ "msg": "User created",

"user": { "ID": 1,

"CreatedAt": "2025-07-13T01:02:05.8064596+05:30",

"UpdatedAt": "2025-07-13T01:02:05.8064596+05:30",

"DeletedAt": null } }

By default, Go only serializes exported fields (those with uppercase first letters)

But GORM doesn't automatically expose fields unless you explicitly add JSON tags

Changing the **model.go** as

package model

import "gorm.io/gorm"

type User struct{

    gorm.Model

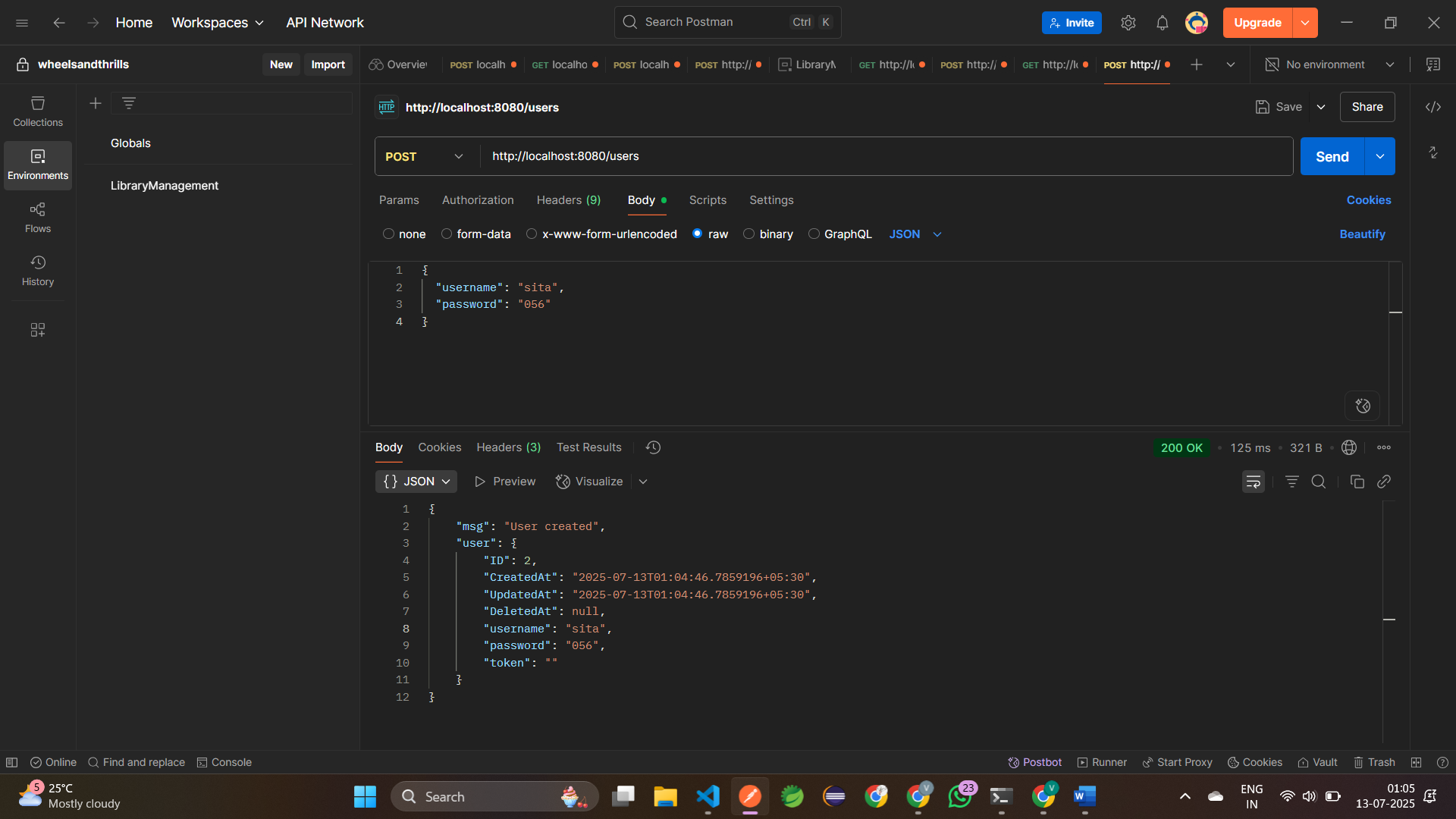
    Username string `json:"username"`

    Password string `json:"password"`

    Token    string `json:"token"`

}

**OUTPUT:** User saved in database

11. Building a login system:

User sends username & password  
 If correct – generate a token, save to database and return it.

If wrong – send an error

* Add **/users/login** endpoint in **main.go**:

router.POST("/users/login", func(c \*gin.Context){

var log struct{

            Username string `json:"username"`

            Password string `json:"password"`

        }

        if e := c.ShouldBindJSON(&log); e!=nil{

            c.JSON(http.StatusBadRequest, gin.H{"error":"Invalid"})

            return

        }

        var user model.User

res := database.DB.Where("username = ? AND password = ?", log.Username, log.Password).First(&user)

        if res.Error != nil{

            c.JSON(http.StatusUnauthorized, gin.H{"error":"Invalid username and password"})

            return

        }

        token := log.Username+"token\_1"

        user.Token=token

        database.DB.Save(&user)

        c.JSON(http.StatusOK,gin.H{

            "Msg":"Login Successful",

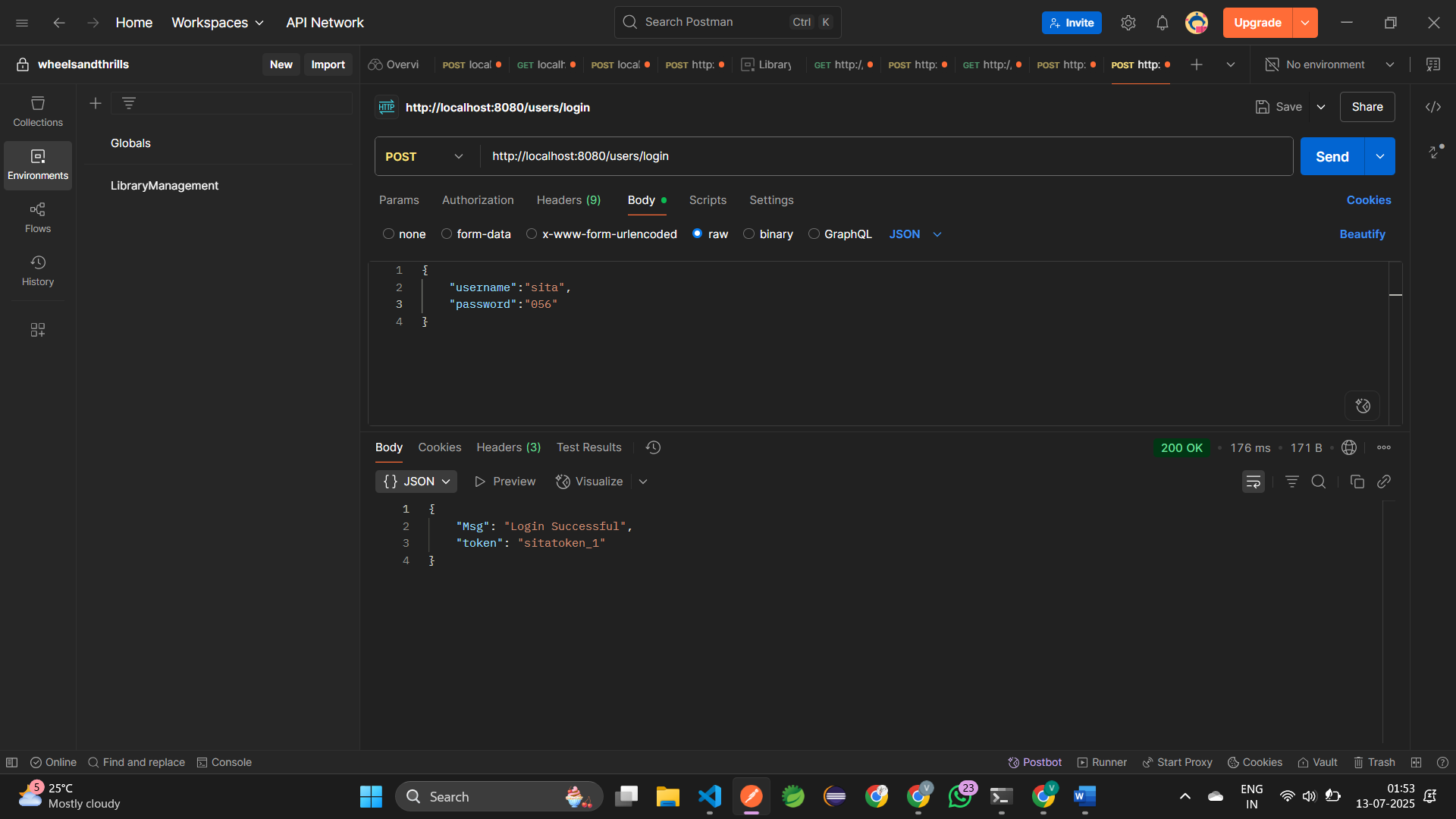
            "token": token,

        })

    })

12. Testing in POSTMAN:

Run the Application : **go run main.go**

URL :[**http://localhost:8080/users/login**](http://localhost:8080/users/login)  
  
13. Creating item Model :

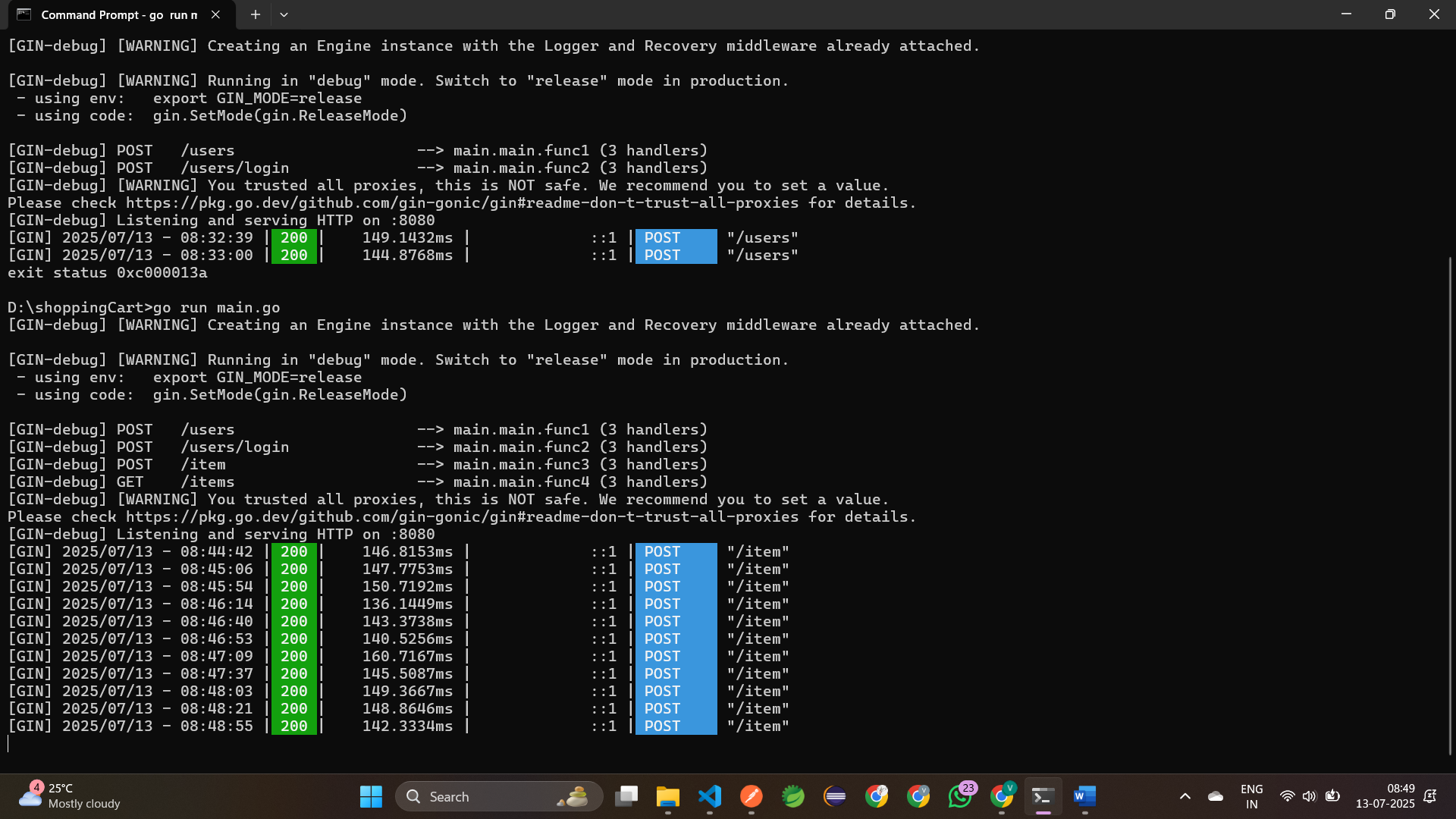
i)navigating to model : **cd model**ii)create an empty file : **type nul > item.go**  
iii)open item.go in vscode and add your products  
  
package model  
import "gorm.io/gorm"  
type Item struct{  
gorm.Model  
Name string `json:"name`  
Status string `json:"status`  
}

iv) change your Init() func in data.go  
 **DB.AutoMigrate(&model.Item{})** – this will create the items table in db.

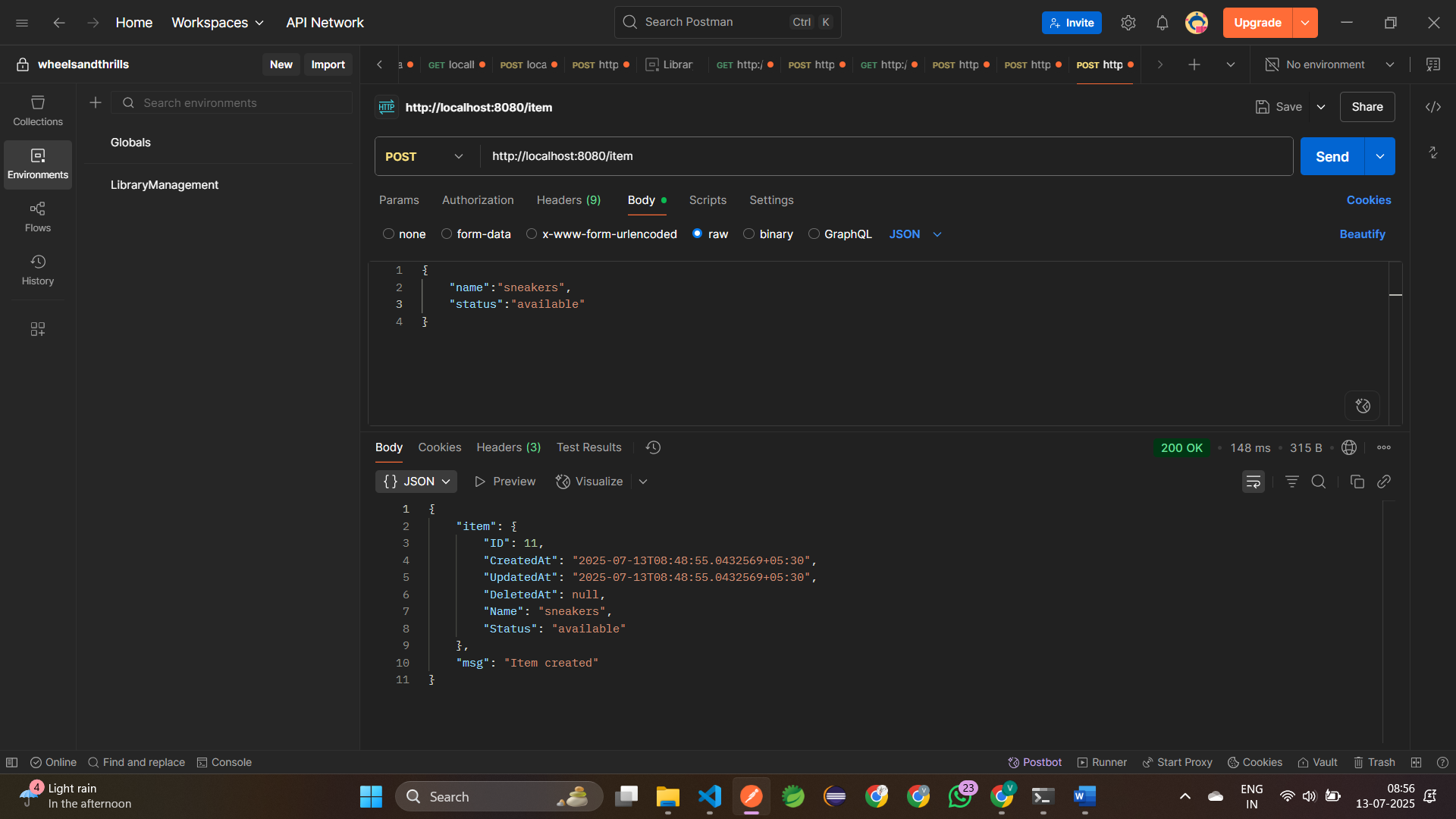
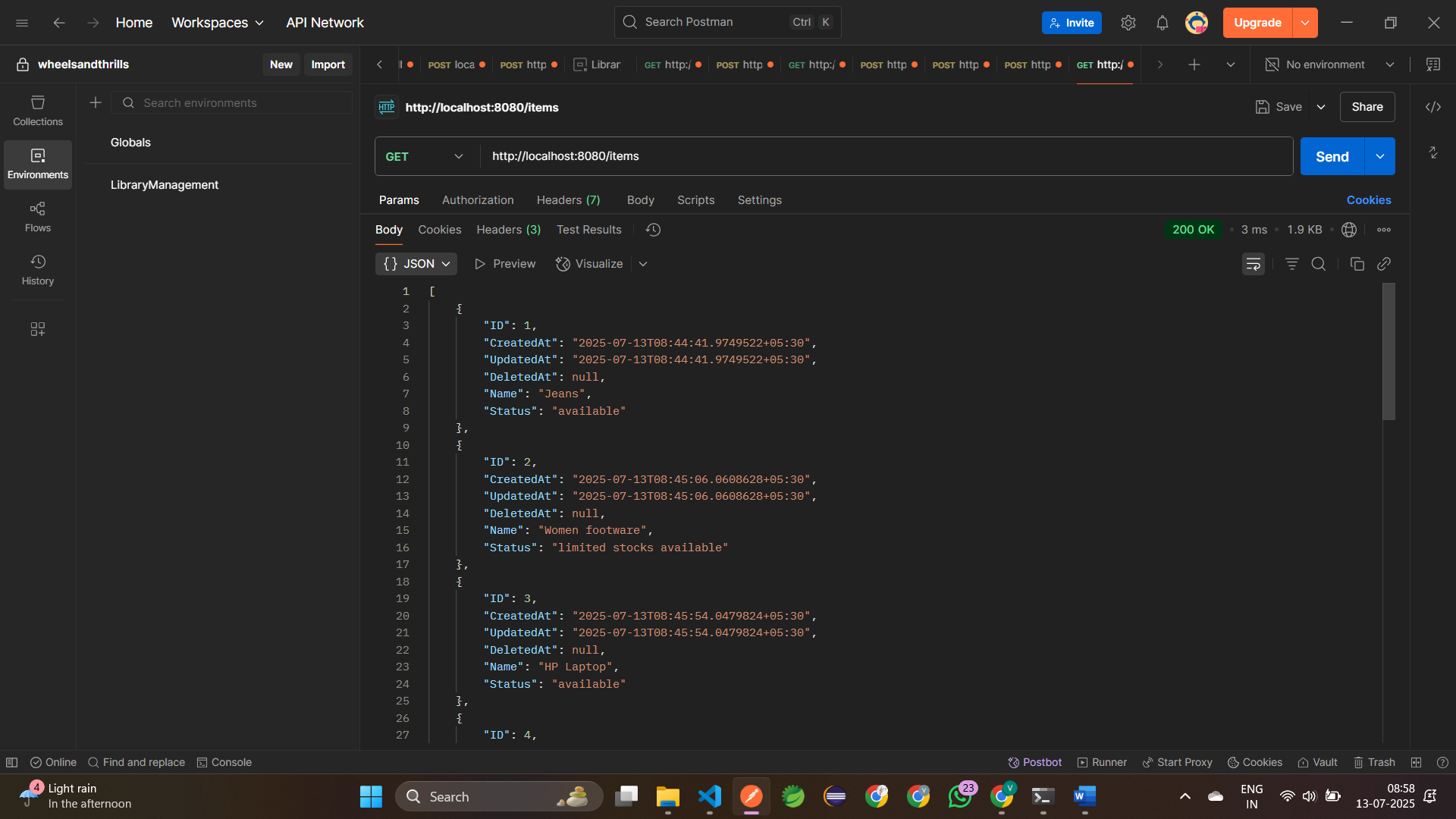
v) creating /item and /items endpoints in main.go:

router.POST("/item",func(c \*gin.Context)  
 var item model.Item  
 if e := c.ShouldBindJSON(&item); e != nil{  
 c.JSON(http.StatusBadRequest, gin.H{"error":"Invalid input"})  
 return  
 }  
 database.DB.Create(&item)  
 c.JSON(http.StatusOK, gin.H{"msg":"Item created","item":item})  
 })  
 //listing all items  
 router.GET("/items",func(c \*gin.Context){  
 var items []model.Item  
 database.DB.Find(&items)  
 c.JSON(http.StatusOK,items)  
 })

14. Testing in POSTMAN:  
 Running the Application : **go run main.go**  
 URL: <http://localhost:8080/item>



Adding items in model:

  
  
  
  
Fetching all the items through GET method : <http://localhost:8080/items>  
  


14. Cart Creation and adding items  
 Every user can have only one cart  
 A cart can contain multiple items  
 Tokens are used to identify who is adding the items  
  
 i)Creating a Cart model:  
 Navigate to model **cd model** and create an empty file **type nul > cart.go** open cart.go in vscode and create the cart struct

package model  
 import "gorm.io/gorm"  
 type Cart struct{  
 gorm.Model  
 UserID uint `json:"user\_id"`  
 Name string `json:"name"`  
 Status string `json:"status"`  
 }

Next create the cartItem in model **type nul > cartItem.go**

package model  
import "gorm.io/gorm"  
type CartItem struct{  
 CartID uint `json:"cartID"`  
 ItemID unit `json:"itemId"`  
}

Auto-migrate both in database/data.go:

**DB.AutoMigrate(&model.Cart{})  
 DB.AutoMigrate(&model.CartItem{})**

Running the Application:

**go run main.go** – this will create and migrate the Cart & CartItem

15. Add to cart API(by token)

When a logged-in user choose an item(by id):  
 1. Read their token  
 2. Find their cart (or create)  
 3. Add item to their cart

router.POST("/carts",func(c \*gin.Context){  
 token := c.GetHeader("Authorization")  
 if token == ""{  
 c.JSON(http.StatusUnauthorized,gin.H{"error":"Missing token"})  
 return  
 } var user model.User  
 if e := database.DB.Where("token = ?",token).First(&user).Error; e != nil{  
 c.JSON(http.StatusUnauthorized,gin.H{"error":"Invalid token"})  
 return  
 } var itemData struct{  
 ItemID uint `json:"item\_id"`  
 Name string `json:"name"`  
 } if e := c.ShouldBindJSON(&itemData); e!=nil{  
 c.JSON(http.StatusBadRequest,gin.H{"error":"Invalid item"})  
 return  
 } //find or create a cart for this user  
 var cart model.Cart  
 if e := database.DB.Where("userID = ?",user.ID).First(&cart).Error; e!=nil{  
 cart=model.Cart{  
 UserID: user.ID,  
 Name: itemData.Name,  
 Status: "active",  
 }  
 database.DB.Create(&cart)  
 } //create a cart item linking item to cart  
 CartItem := model.CartItem{  
 CartID: cart.ID,  
 ItemID: itemData.ItemID,  
 }  
 database.DB.Create(&CartItem)  
 c.JSON(http.StatusOK,gin.H{  
 "Msg":"Item added to cart",  
 "cart": cart.ID,  
 })  
 })

4. Testing in POSTMAN:

**URL** : <http://localhost:8080/carts>

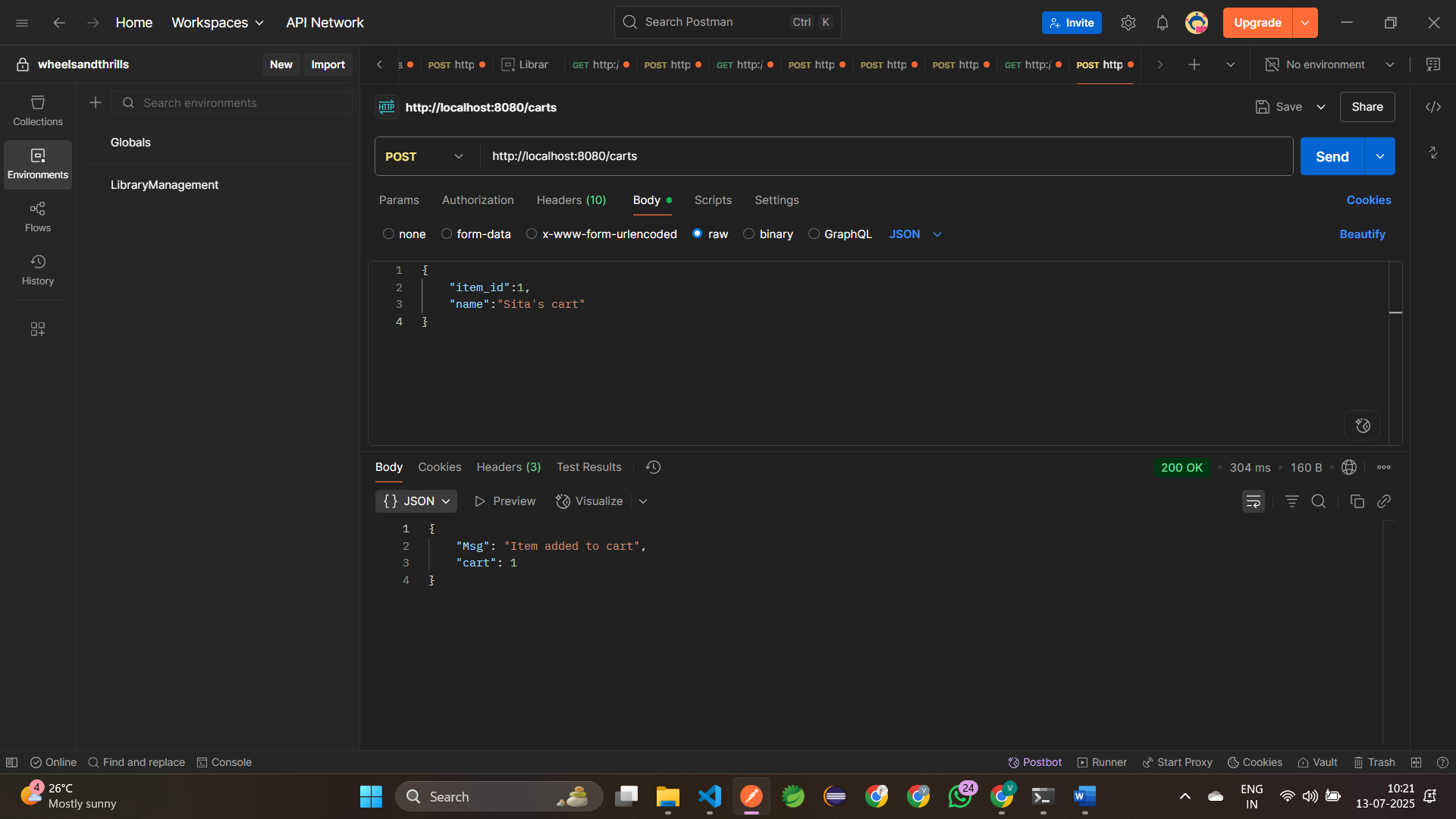
**Header:**  
 Key: Authorization

Value: sitatoken\_1

**Add a json body:**

{  
 "item\_id":1,  
 "name":"Sita's cart"  
 }

OUTPUT:



16. View cart items:

Add GET/carts endpoint in **main.go**

router.GET("/carts",func(c \*gin.Context){  
 token := c.GetHeader("Authorization")  
 if token == ""{  
 c.JSON(http.StatusUnauthorized,gin.H{"error":"Missing token"})  
 return  
 }  
 var user model.User  
 if e := database.DB.Where("token = ?",token).First(&user).Error; e != nil{  
 c.JSON(http.StatusUnauthorized,gin.H{"error":"Invalid"})  
 return  
 }  
 var cart model.Cart  
 if e := database.DB.Where("user\_id = ?",user.ID).First(&cart).Error; e != nil{  
 c.JSON(http.StatusNotFound,gin.H{"error":"Cart not found"})  
 return  
 }

var cartItems []model.CartItem  
 database.DB.Where("cart\_id = ?",cart.ID).Find(&cartItems)          
 c.JSON(http.StatusOK,gin.H{  
 "cart":cart,  
 "item":cartItems,  
 })  
 })

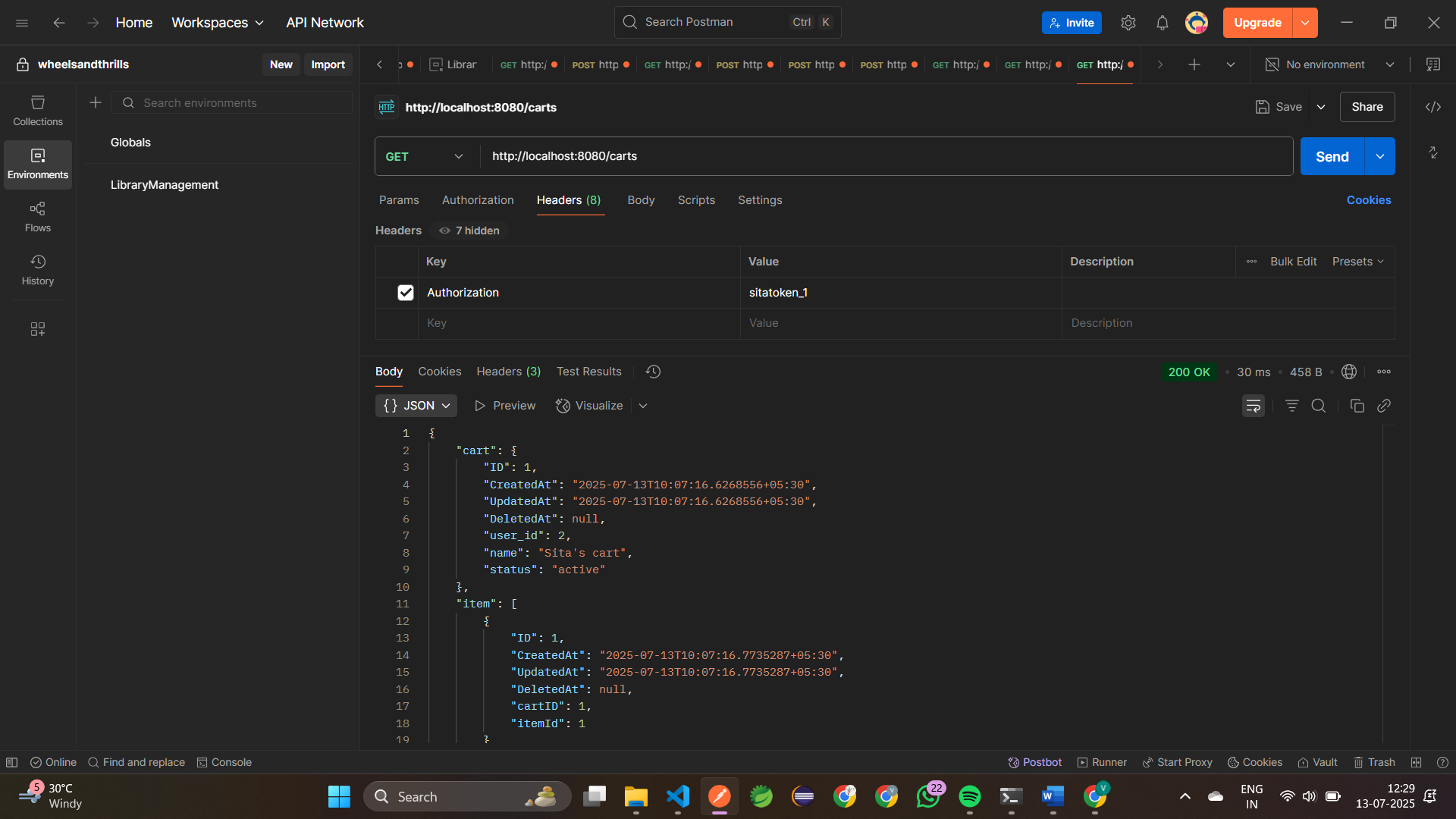
**Testing in POSTMAN:**   
 Endpoint : GET/carts

Header:

Authorization: sitatoken\_1

Logic:

i)Read token  
 ii)Find the user by ID  
 iii)Find their active cart  
 iv)Fetch items from the cart



17. Placing an Order:

User sends a request to place an order  
Validating the user through token  
Fetch the cart  
Create a new order  
Clear or deactivate the cart

i) Navigate to the model **cd model** and create an empty file **type nul > order.go**

ii) Open **order.go** in vscode and create a **Order Struct**

package model  
 import "gorm.io/gorm"  
 type Order struct{  
 gorm.Model  
 UserID uint   
 CartID uint  
 Status string   
 }

iii) Auto-migrate both in database/data.go:

**DB.AutoMigrate(&model.Order{})**

iv) Add POST/orders endpoint in main.go:

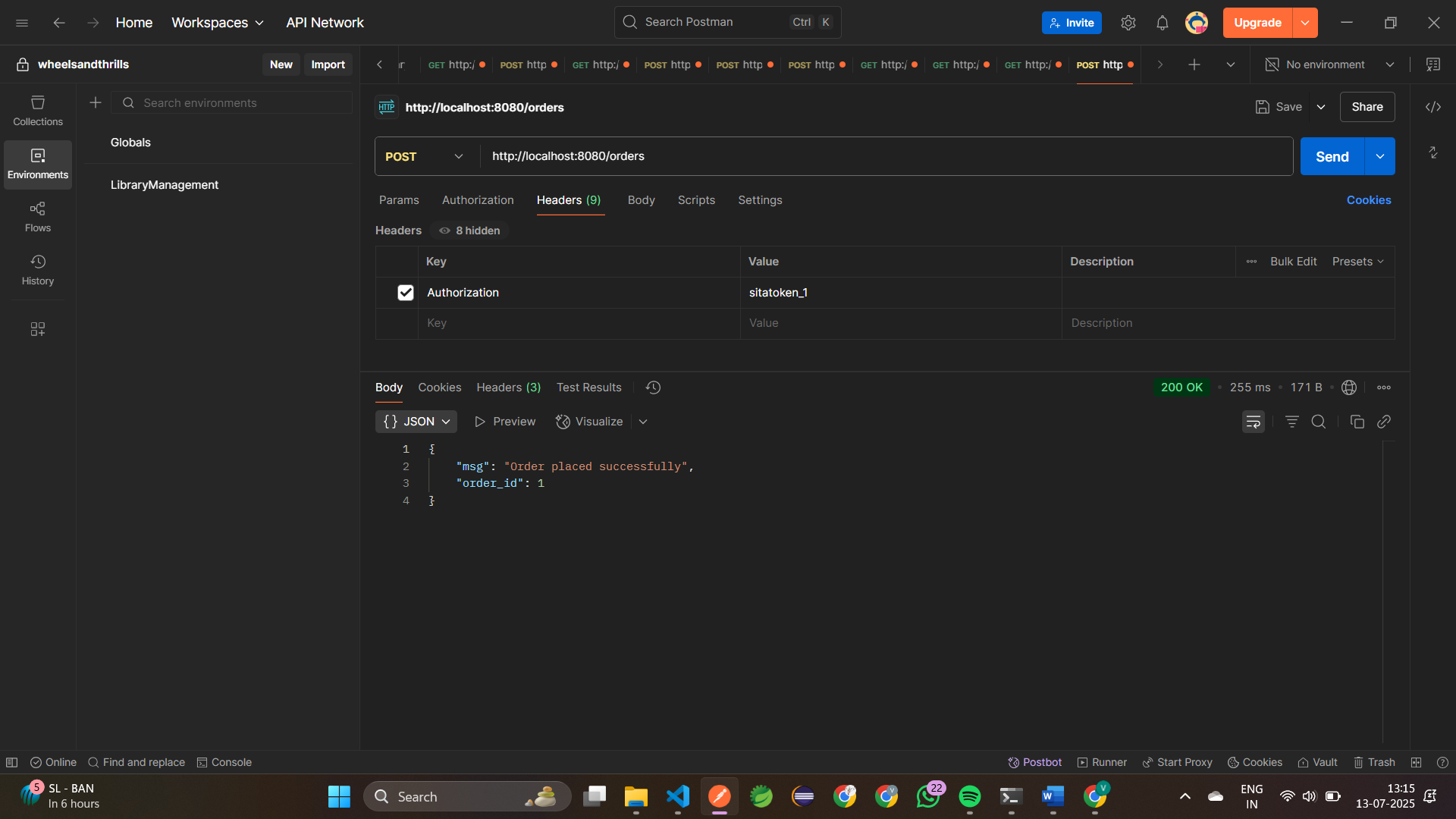
router.POST("/orders",func(c \*gin.Context){  
        token := c.GetHeader("Authorization")  
        if token == ""{  
             c.JSON(http.StatusUnauthorized,gin.H{"error":"Missing token"})  
             return  
         }  
         var user model.User  
         if e := database.DB.Where("token = ?",token).First(&user).Error; e != nil{  
             c.JSON(http.StatusUnauthorized,gin.H{"error":"Invalid token"})  
             return  
         }  
         var cart model.Cart  
         if e := database.DB.Where("user\_id = ?",user.ID).First(&cart).Error; e != nil{  
             c.JSON(http.StatusNotFound,gin.H{"error":"Cart not found"})  
             return  
         }  
         order := model.Order{  
            UserID: user.ID,  
            CartID: cart.ID,  
            Status: "Placed",  
         }  
         database.DB.Create(&order)  
         cart.Status = "inactive"  
         database.DB.Save(&cart)  
         c.JSON(http.StatusOK,gin.H{"msg":"Order placed successfully","order\_id":order.ID,})

    })

v) Run the Application: **go run main.go**

vi) Testing in POSTMAN:

URL : <http://localhost:8080/orders>

Headers :   
 Key: Authorization  
 Value: sitatoken\_1  
  
  
  
18. View Orders:

i)Add GET/orders endpoint in **main.go** router.GET("/orders",func(c \*gin.Context){  
         token := c.GetHeader("Authorization")  
         if token == ""{  
             c.JSON(http.StatusUnauthorized,gin.H{"error":"Missing token"})  
             return  
         }  
         var user model.User  
         if e := database.DB.Where("token = ?",token).First(&user).Error; e != nil{  
             c.JSON(http.StatusUnauthorized,gin.H{"error":"Invalid token"})  
             return  
         }  
         var orders []model.Order  
         database.DB.Where("user\_id = ?",user.ID).Find(&orders)  
         c.JSON(http.StatusOK,gin.H{  
             "orders":orders,  
         })

    })

**ii)Testing in POSTMAN:**  
 Endpoint : GET/carts  
 Header:

Authorization: sitatoken\_1  
  
