Delivery App Simulator

Acknowledgement

We would like to express our sincere gratitude to everyone who supported us throughout the duration of this project. As a team of two, we especially acknowledge each other Lakshmi Priya, Khajabi for our dedication, collaboration, and joint contributions toward the successfully completion of this project.

Project Guide

V. Sai Chandana

FAANG CEO

J.Venkat

Technology Stack: Java, Spring Boot, Maven, Rest APIs, Git

INTRODUCTION:

The **Delivery App Simulator** is a Spring Boot–based application designed to emulate the core functionality of a real-world delivery platform. It enables customers to place orders, track delivery status, and view order history while ensuring accurate order processing and delivery management.

The system leverages **RESTful APIs** for modular and scalable backend design, enabling easy integration with frontend applications or third-party services. For data persistence, it supports **MongoDB** or relational databases, storing detailed order and delivery information for quick retrieval and historical analysis.

For development workflow, the project uses **Git for version control**, allowing collaborative development, code tracking, and rollback if necessary. **Postman collections** are included for API testing, ensuring that endpoints function correctly and meet expected requirements.

OBJECTIVES:

Customer Order Management:

Enable customers to easily place, update, and cancel delivery orders through a user-friendly interface. The system ensures smooth order handling and efficient processing.

Real-Time Delivery Tracking:

Provide live updates on the status of deliveries(PENDING, IN_TRANSIT, DELIVERED) so that customers and administrators can monitor orders at any time.

Order and Delivery History:

Maintain a comprehensive history of all orders and deliveries, allowing users to review past transactions and track repeat orders.

Modular API Design with Spring Boot:

Use Spring Boot to build modular, scalable, and maintainable APIs. The architecture follows clean coding principles and ensures easy integration with other systems.

Development and Testing Support:

Leverage Git for version control and Postman collections for testing APIs. This ensures collaborative development, efficient debugging, and reliable deployment.

KEY FEATURES:

- **Order Management** – Place, update, cancel, and view orders.
- **Delivery Tracking** – Track delivery status in real time.
- **Item Management** – Add, update, and fetch delivery items with pricing.
- **Order History** – Retrieve complete order history for any customer.
- **Modular API Design** – Built with Spring Boot REST APIs following OOP principles.
- **Database Persistence** – Store and retrieve orders, deliveries, and items reliably.
- **Development Tools** – Uses Maven for build lifecycle, Git for version control, and Postman for testing.

TOOLS AND TECHNOLOGIES:

SpringBoot :

Used for building the RESTful backend API with modular, scalable architecture.

MongoDB / Relational Database:

Stores order, delivery, and user information with efficient retrieval and query support.

Postman:

For API testing and validation of endpoints, including GET, PUT, and POST requests.

Git :

Version control system for code management,collaboration, and maintaining project history.

Maven / Gradle :

Build tools for dependency management and project automation.

Spring Data JPA / Spring Data MongoDB:

Simplifies data access and repository implementation.

Code Implementation:

Entity class:

Item.java:

```
package com.delivery.entity;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
@Document(collection = "items")
public class Item {
   @Id
   private String id;
   private String name;
   private Double price;
   public Item() {  }
        public Item(String id, String name, Double price) {
                super();
                this.id = id;
                this.name = name;
                this.price = price;
        }
        public String getId() {  return id;  }
        public void setId(String id) {  this.id = id; }
        public String getName() {      return name;    }
        public void setName(String name) {  this.name = name;      }
        public Double getPrice() {     return price;    }
        public void setPrice(Double price) {  this.price = price;                }
   }
```

Order.java:

```
package com.delivery.entity;
import java.time.LocalDateTime;
import java.util.List;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import com.fasterxml.jackson.annotation.JsonProperty;
@Document(collection = "orders")
public class Order {
    @Id
    private String id;
    @JsonProperty("userId")
    private String customerId;
    private String address;
    private Double price;
    private String status;  // NEW, PROCESSING, DELIVERED, CANCELLED
    private LocalDateTime createdAt = LocalDateTime.now();
    private LocalDateTime updatedAt = LocalDateTime.now();
    private List<String> itemIds;

    public Order() {}

    public List<String> getItemIds() { return itemIds; }
    public void setItemIds(List<String> itemIds) { this.itemIds = itemIds; }
    public String getId() { return id; }
    public void setId(String id) { this.id = id; }
    public String getCustomerId() { return customerId; }
    public void setCustomerId(String customerId) { this.customerId = customerId; }
     public String getAddress() { return address; }
```

```
    public void setAddress(String address) { this.address = address; }
     public Double getPrice() { return price; }
    public void setPrice(Double price) { this.price = price; }
    public String getStatus() { return status; }
    public void setStatus(String status) { this.status = status; }
    public LocalDateTime getCreatedAt() { return createdAt; }
    public void setCreatedAt(LocalDateTime createdAt) { this.createdAt = createdAt; }
    public LocalDateTime getUpdatedAt() { return updatedAt; }
    public void setUpdatedAt(LocalDateTime updatedAt) { this.updatedAt = updatedAt; }
}
```

Delivery.java

```
package com.delivery.entity;
import java.time.LocalDateTime;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
@Document(collection = "deliveries")
public class Delivery {
    @Id
    private String id;
    private String orderId;
    private String status; // PENDING, IN_TRANSIT, DELIVERED
    private LocalDateTime createdAt = LocalDateTime.now();
    private LocalDateTime updatedAt = LocalDateTime.now();

    public Delivery() {    }
        public Delivery(String id, String orderId, String status, LocalDateTime createdAt,
LocalDateTime updatedAt) {
                this.id = id;
                this.orderId = orderId;
                this.status = status;
```

```
        this.createdAt = createdAt;
        this.updatedAt = updatedAt;   }
    public String getId() {          return id;    }
    public void setId(String id) { this.id = id;     }
    public String getOrderId() { return orderId; }
    public void setOrderId(String orderId) {  this.orderId = orderId;  }
    public String getStatus() {     return status;  }
    public void setStatus(String status) {this.status = status;     }
    public LocalDateTime getCreatedAt() {       return createdAt;       }
    public void setCreatedAt(LocalDateTime createdAt) {
        this.createdAt = createdAt;  }
    public LocalDateTime getUpdatedAt() {      return updatedAt;      }
    public void setUpdatedAt(LocalDateTime updatedAt) {     this.updatedAt =
updatedAt;
    }
}
```

Repository Layer:

ItemRepository.java:

```
package com.delivery.repository;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
import com.delivery.entity.Item;

@Repository
public interface ItemRepository extends MongoRepository<Item,String>{

}
```

OrderRepository.java:

```
package com.delivery.repository;
import java.util.List;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
import com.delivery.entity.Order;

@Repository
public interface OrderRepository extends MongoRepository<Order, String>{
        List<Order> findByCustomerId(String customerId);

}
```

DeliveryRepository.java:

```
package com.delivery.repository;
import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
import com.delivery.entity.Delivery;
@Repository
public interface DeliveryRepository extends MongoRepository<Delivery, String> {
   Delivery findByOrderId(String orderId);
}
```

Service Layer:

ItemService:

```
package com.delivery.service;
import java.util.List;
import org.springframework.stereotype.Service;
import com.delivery.entity.Item;
import com.delivery.repository.ItemRepository;

@Service
public class ItemService {
   private final ItemRepository itemRepo;
   public ItemService(ItemRepository itemRepo) {
         this.itemRepo = itemRepo;     }
   public List<Item> getAllItems() {
      return itemRepo.findAll();
   }
   public Item addItem(Item item) {
      return itemRepo.save(item);
   }
   public Item updateItem(String id, Item updated) {
                  if(itemRepo.findById(id) != null) {
                        System.out.println("Item found");
                  }else {
                        System.out.println("Item not found::");
                  }
      return itemRepo.save(updated);
   }
}
```

OrderService.java:

```
package com.delivery.service;
import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;
import org.springframework.stereotype.Service;
import com.delivery.entity.Item;
import com.delivery.entity.Order;
import com.delivery.repository.ItemRepository;
import com.delivery.repository.OrderRepository;
@Service
public class OrderService {
    private final OrderRepository orderRepo;
    private final ItemRepository itemRepo;
    public OrderService(OrderRepository orderRepo, ItemRepository itemRepo) {
        this.orderRepo = orderRepo;
        this.itemRepo = itemRepo;
    }
    public Order placeOrder(Order order) {
        double totalPrice = 0.0;
        if (order.getItemIds() != null && !order.getItemIds().isEmpty()) {
            for (int i = 0; i < order.getItemIds().size(); i++) {
                String itemId = order.getItemIds().get(i);
                Optional<Item> itemOpt = itemRepo.findById(itemId);
                if (itemOpt.isPresent()) {
                    Item item = itemOpt.get();
                    if (item.getPrice() != null) {
                            totalPrice += item.getPrice();
```

```
            } else {
                System.out.println("⚠ Item price is null for: " + itemId);
            }
        } else {
            System.out.println("⚠ Item not found: " + itemId);
        }
    }
} else {
    System.out.println("⚠ No item IDs provided!");
}
order.setPrice(totalPrice);
order.setStatus("NEW");
order.setCreatedAt(LocalDateTime.now());
order.setUpdatedAt(LocalDateTime.now());
return orderRepo.save(order);
}
public Order updateOrder(String id, Order updated) {
Optional<Order> existingOpt = orderRepo.findById(id);
if (!existingOpt.isPresent()) {
    throw new RuntimeException("Order not found: " + id);
}
Order existing = existingOpt.get();
if (updated.getAddress() != null) {
    existing.setAddress(updated.getAddress());
}
if (updated.getItemIds() != null && !updated.getItemIds().isEmpty()) {
    existing.setItemIds(updated.getItemIds());
    double totalPrice = 0.0;
    for (int i = 0; i < updated.getItemIds().size(); i++) {
        String itemId = updated.getItemIds().get(i);
```

```
            Optional<Item> itemOpt = itemRepo.findById(itemId);
            if (itemOpt.isPresent()) {
                Item item = itemOpt.get();
               if (item.getPrice() != null) {
                   totalPrice += item.getPrice();
               } else {
                   System.out.println("⚠ Item price is null for: " + itemId);
               }
            } else {
               System.out.println("⚠ Item not found: " + itemId);
            }
         }
         existing.setPrice(totalPrice);
      }
      existing.setUpdatedAt(LocalDateTime.now());
      return orderRepo.save(existing);
   }
   public void cancelOrder(String id) {
      orderRepo.deleteById(id);
   }
   public List<Order> getOrder(String id) {
      return orderRepo.findAll();
   }
   public List<Order> getHistory(String customerId) {
      return orderRepo.findByCustomerId(customerId);
   }
}
```

DeliveryService:

```
package com.delivery.service;
import java.time.LocalDateTime;
import org.springframework.stereotype.Service;
import com.delivery.entity.Delivery;
import com.delivery.repository.DeliveryRepository;
@Service
public class DeliveryService {
    private final DeliveryRepository deliveryRepo;
    public DeliveryService(DeliveryRepository deliveryRepo) {
        this.deliveryRepo = deliveryRepo;
    }
    public Delivery getStatus(String orderId) {
        return deliveryRepo.findByOrderId(orderId);
    }
    public Delivery updateStatus(String orderId, String status) {
        Delivery delivery = deliveryRepo.findByOrderId(orderId);
        if (delivery == null) {
            delivery = new Delivery();
            delivery.setOrderId(orderId);
        }
        delivery.setStatus(status);
        delivery.setUpdatedAt(LocalDateTime.now());
        return deliveryRepo.save(delivery);
    }
}
```

Controller Layer:

ItemController.java:

```
package com.delivery.controller;
import java.util.List;
import java.util.Map;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.delivery.entity.Item;
import com.delivery.service.ItemService;
@RestController
@RequestMapping("/items")
public class ItemController {
    private final ItemService itemService;
    public ItemController(ItemService itemService) {
        this.itemService = itemService;
    }
    @GetMapping
    public List<Item> getItems() {
        return itemService.getAllItems();
    }
    @PostMapping("/add")
    public ResponseEntity<Object> addItem(@RequestBody Item item) {
```

```
        Item saved = itemService.addItem(item);
        return ResponseEntity.ok(Map.of("message", "Item added successfully", "data", saved));
    }
    @PutMapping("/{id}")
    public ResponseEntity<Object> updateItem(@PathVariable String id, @RequestBody Item
item) {
        Item updated = itemService.updateItem(id, item);
        return ResponseEntity.ok(Map.of("message", "Item updated successfully", "data",
updated));
    }
}
```

OrderController.java:

```
package com.delivery.controller;
import java.util.List;
import java.util.Map;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.delivery.entity.Order;
import com.delivery.service.OrderService;
@RestController
@RequestMapping("/orders")
public class OrderController {
    private final OrderService orderService;
    public OrderController(OrderService orderService) {
```

```
        this.orderService = orderService;
    }
    @PostMapping("/add")
    public ResponseEntity<Object> placeOrder(@RequestBody Order order) {
        Order saved = orderService.placeOrder(order);
        return ResponseEntity.ok(Map.of("message", "Order placed successfully", "data",
saved));
    }
    @GetMapping("/{id}")
    public List<Order> getOrder(@PathVariable String id) {
        return orderService.getOrder(id);
    }
    @PutMapping("/{id}")
    public Order updateOrder(@PathVariable String id, @RequestBody Order order) {
        return orderService.updateOrder(id, order);
    }
    @DeleteMapping("/{id}")
    public void cancelOrder(@PathVariable String id) {
        orderService.cancelOrder(id);
    }
    @GetMapping("/history/{userId}")
    public List<Order> getHistory(@PathVariable String userId) {
        return orderService.getHistory(userId);
    }
}
```

DeliveryController.java:

```
package com.delivery.controller;
import java.util.Map;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
```

```
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import com.delivery.entity.Delivery;
import com.delivery.service.DeliveryService;
@RestController
@RequestMapping("/delivery")
public class DeliveryController {
    private final DeliveryService deliveryService;
    public DeliveryController(DeliveryService deliveryService) {
        this.deliveryService = deliveryService;
    }
    @GetMapping("/{orderId}/status")
    public ResponseEntity<Object> getStatus(@PathVariable String orderId) {
        Delivery deliver = deliveryService.getStatus(orderId);
        return ResponseEntity.ok(Map.of(
            "message", "Delivery status fetched successfully",
            "data", deliver
        ));
    }
    @PutMapping("/{orderId}/cancel")
    public ResponseEntity<Object> cancelOrder(@PathVariable String orderId) {
        Delivery deliver = deliveryService.getStatus(orderId);
        if (deliver == null) {
            return ResponseEntity.notFound().build();
        }
        String status = deliver.getStatus() != null ? deliver.getStatus().toUpperCase() : "";
        if ("DELIVERED".equals(status)) {
            return ResponseEntity
```

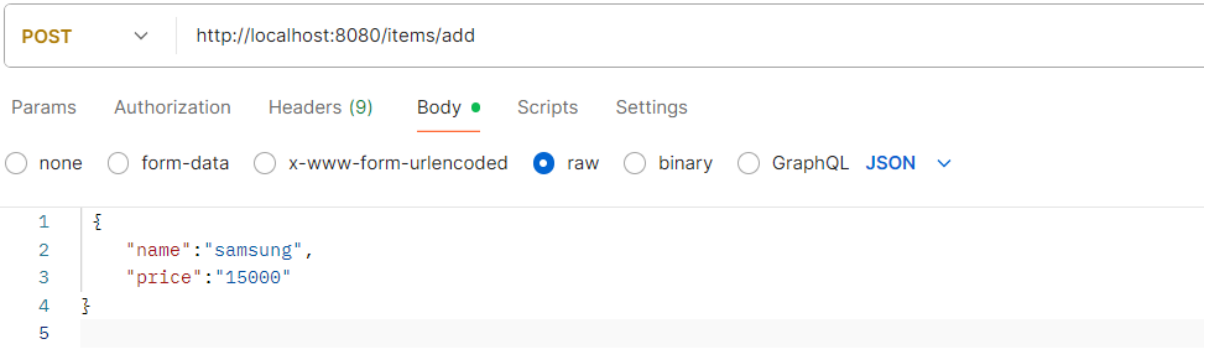
```
                .badRequest()
                .body(Map.of("message", "Order already delivered, cannot cancel", "data",
deliver));
        }
        if ("CANCELLED".equals(status)) {
            return ResponseEntity
                .badRequest()
                .body(Map.of("message", "Order already cancelled", "data", deliver));
        }
        Delivery updated = deliveryService.updateStatus(orderId, "CANCELLED");
        return ResponseEntity.ok(Map.of("message", "Order cancelled successfully", "data",
updated));
    }
@PutMapping("/{orderId}/update")
    public ResponseEntity<Object> updateStatus(@PathVariable String orderId,
     @RequestBody Delivery request) {
        Delivery updated = deliveryService.updateStatus(orderId, request.getStatus());
        return ResponseEntity.ok(Map.of(
            "message", "Delivery status updated successfully",
            "data", updated
        ));
    }
```

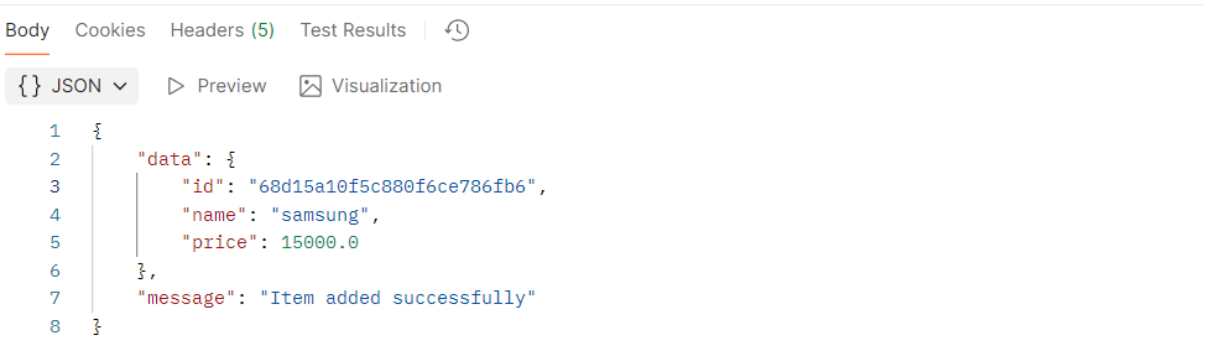
OutPut Format:

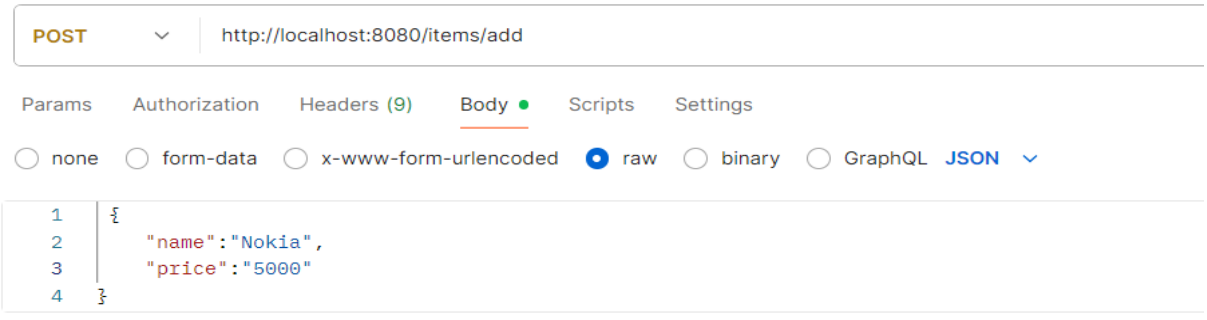
Items List:

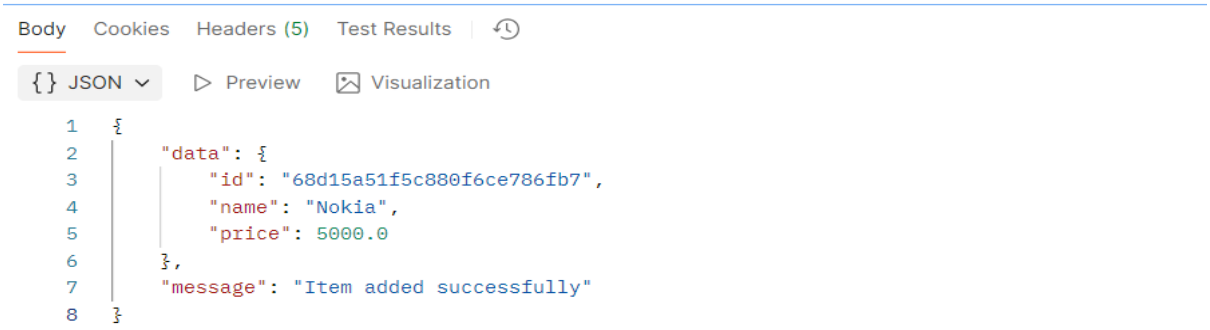
1.POST Request (Add New Item):

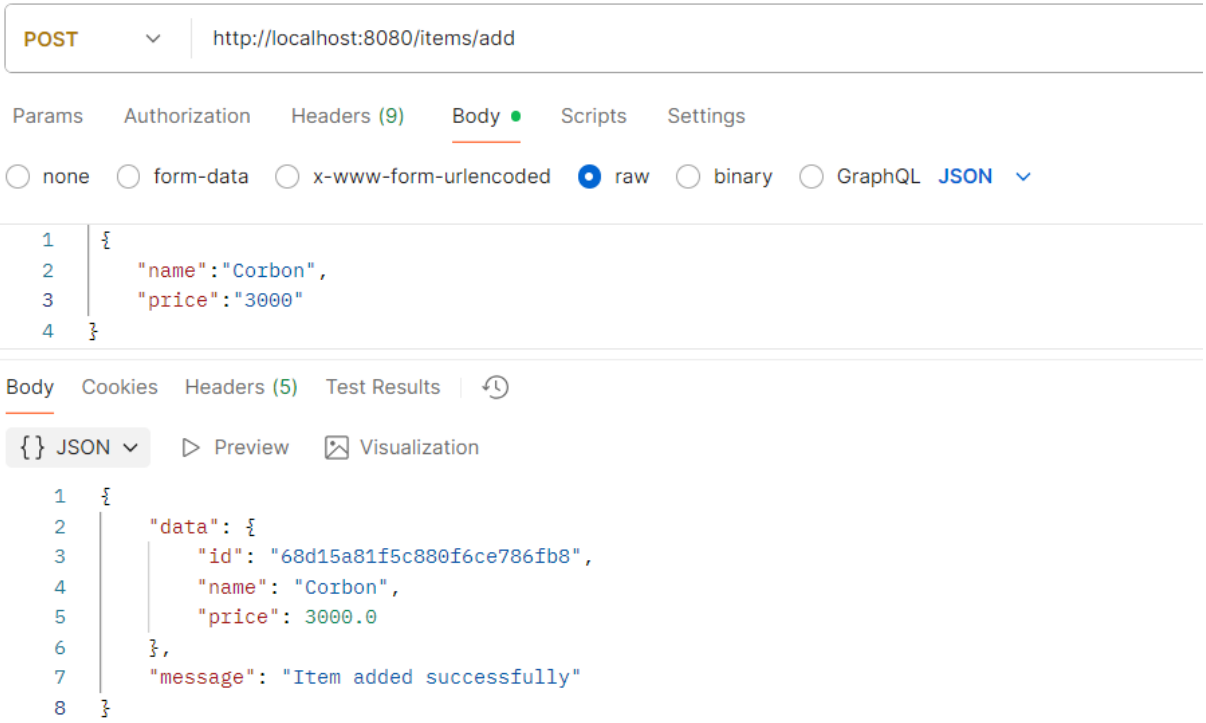
- **Method**: POST
- **URL**: http://localhost:8080/items/add





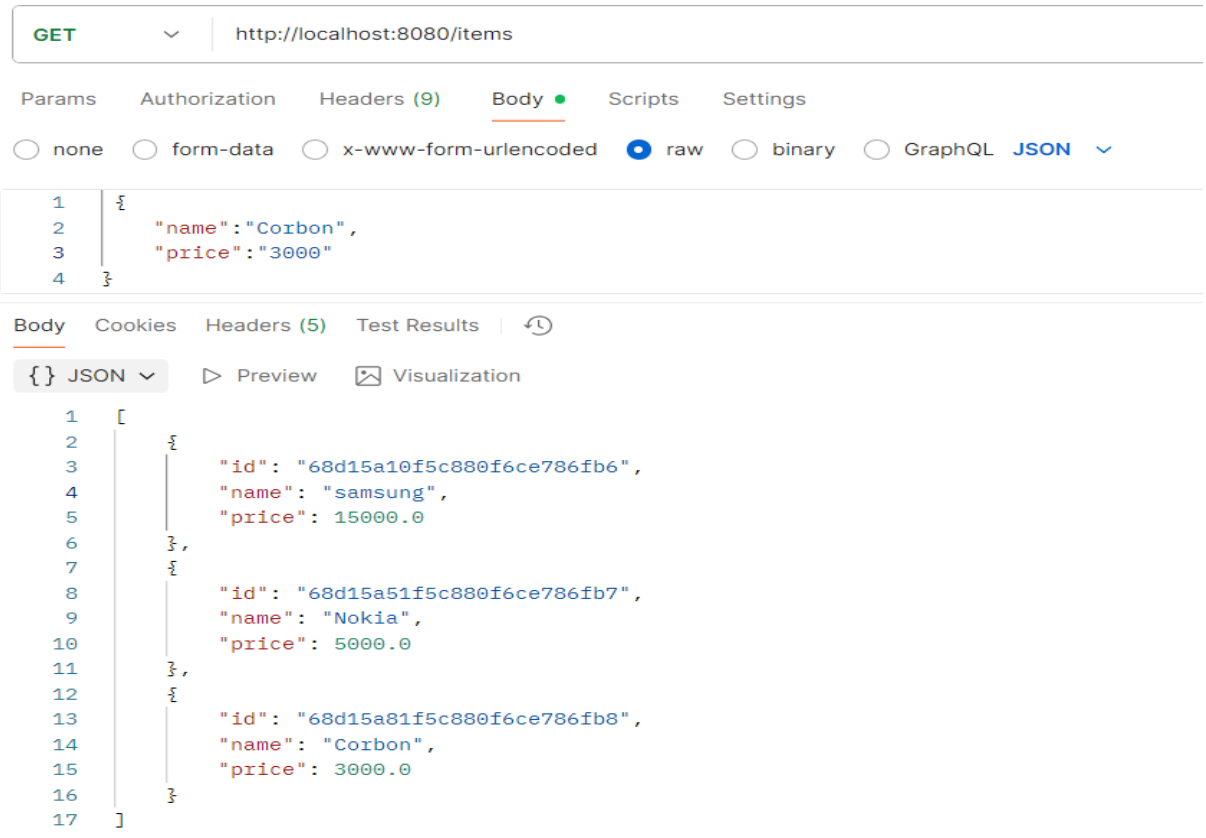






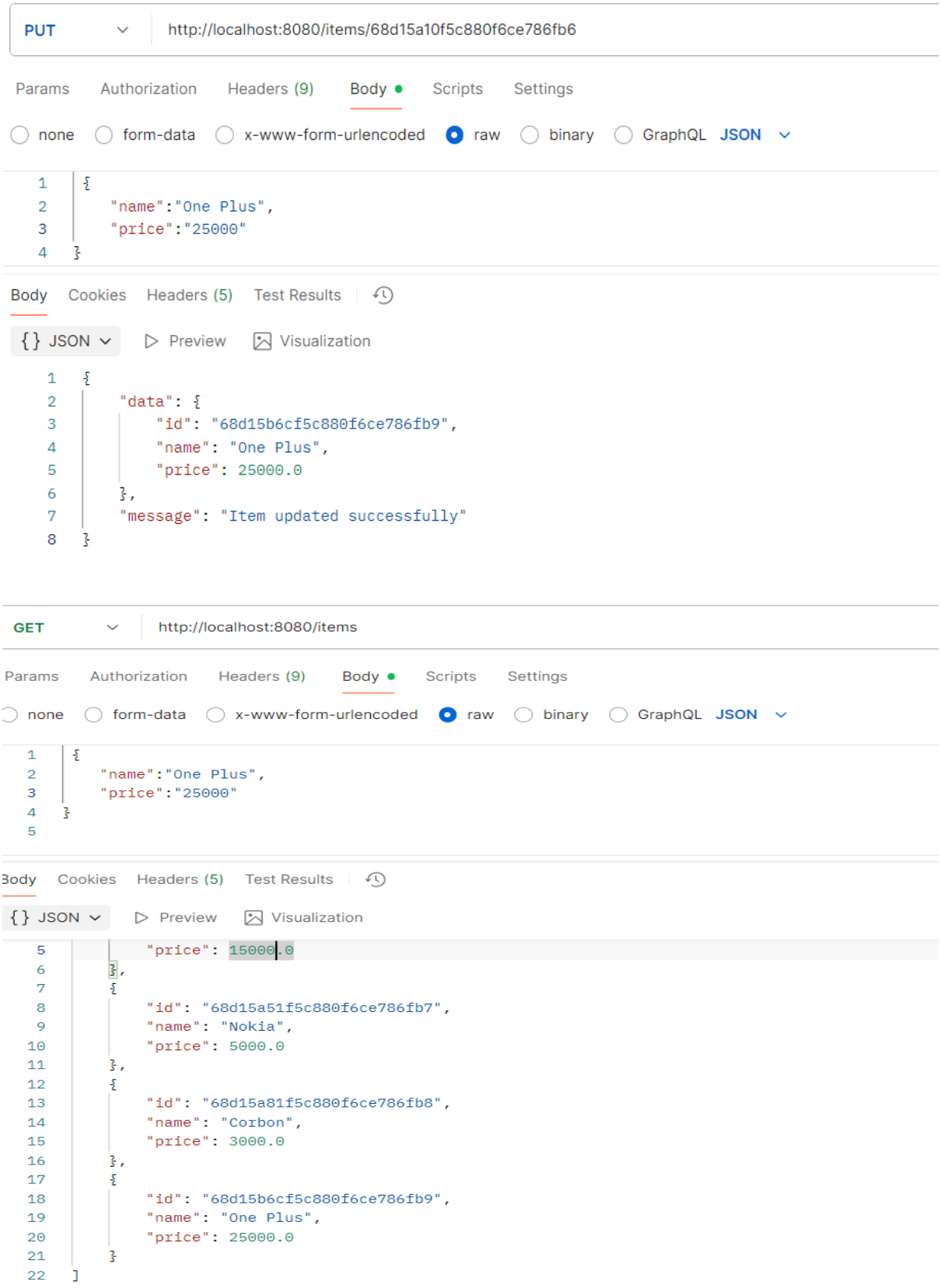
2. GET Request (Retrieve All Items)

- **Method**: GET
- **URL**: http://localhost:8080/items



3. PUT Request (Update Existing Item)

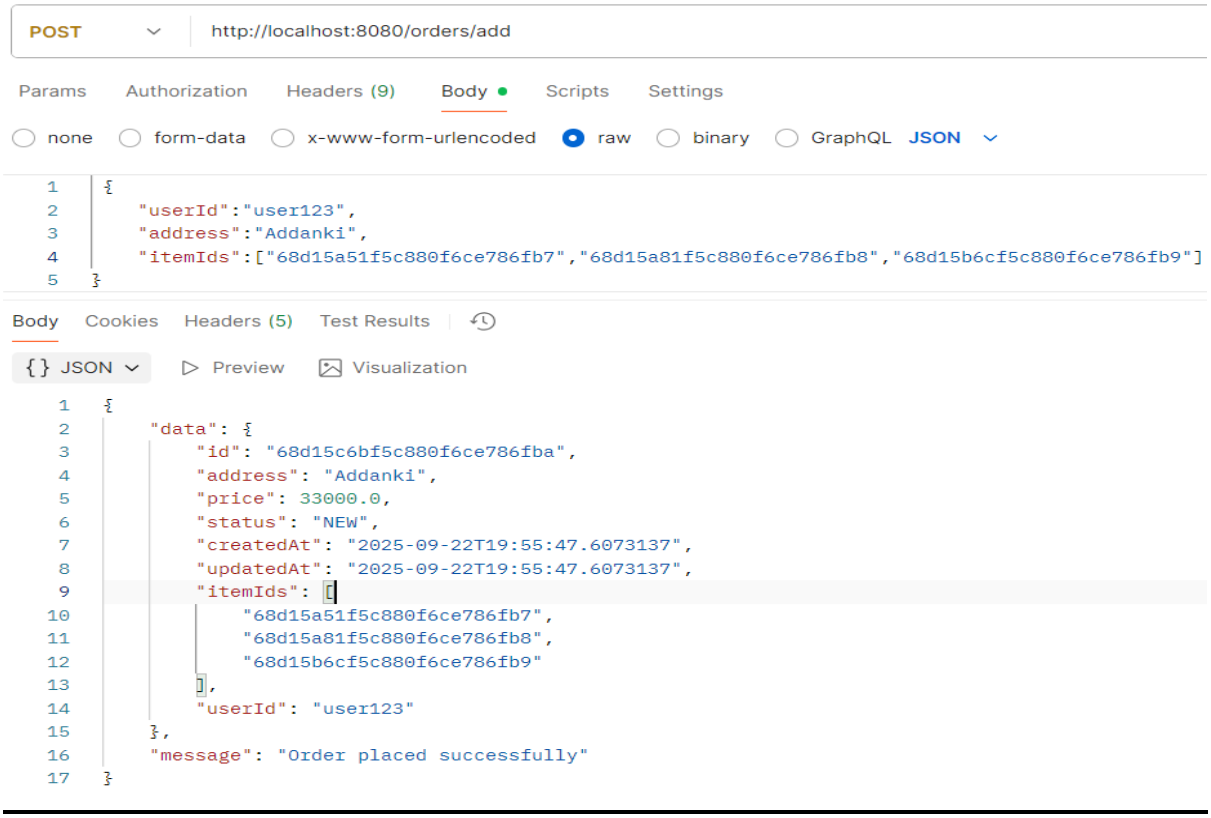
- **Method**: PUT
- **URL**: http://localhost:8080/items/{id}

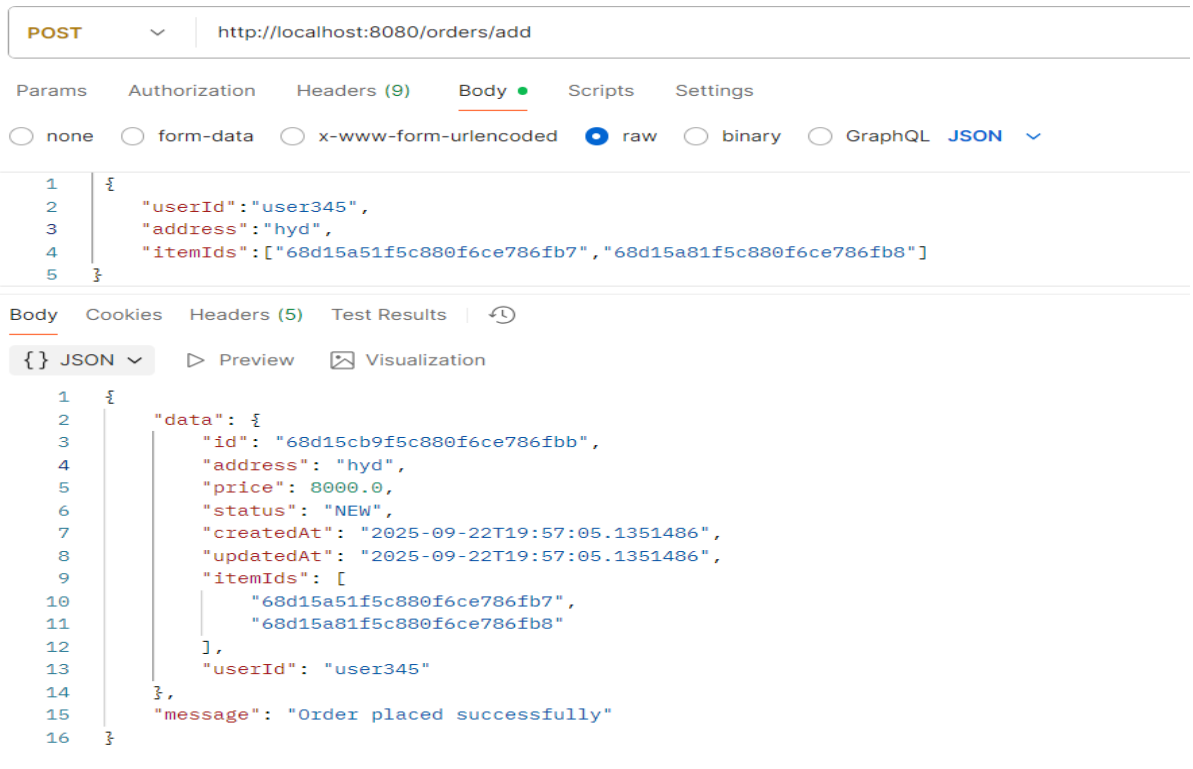


Order List:

1.POST Request (Add New Item):

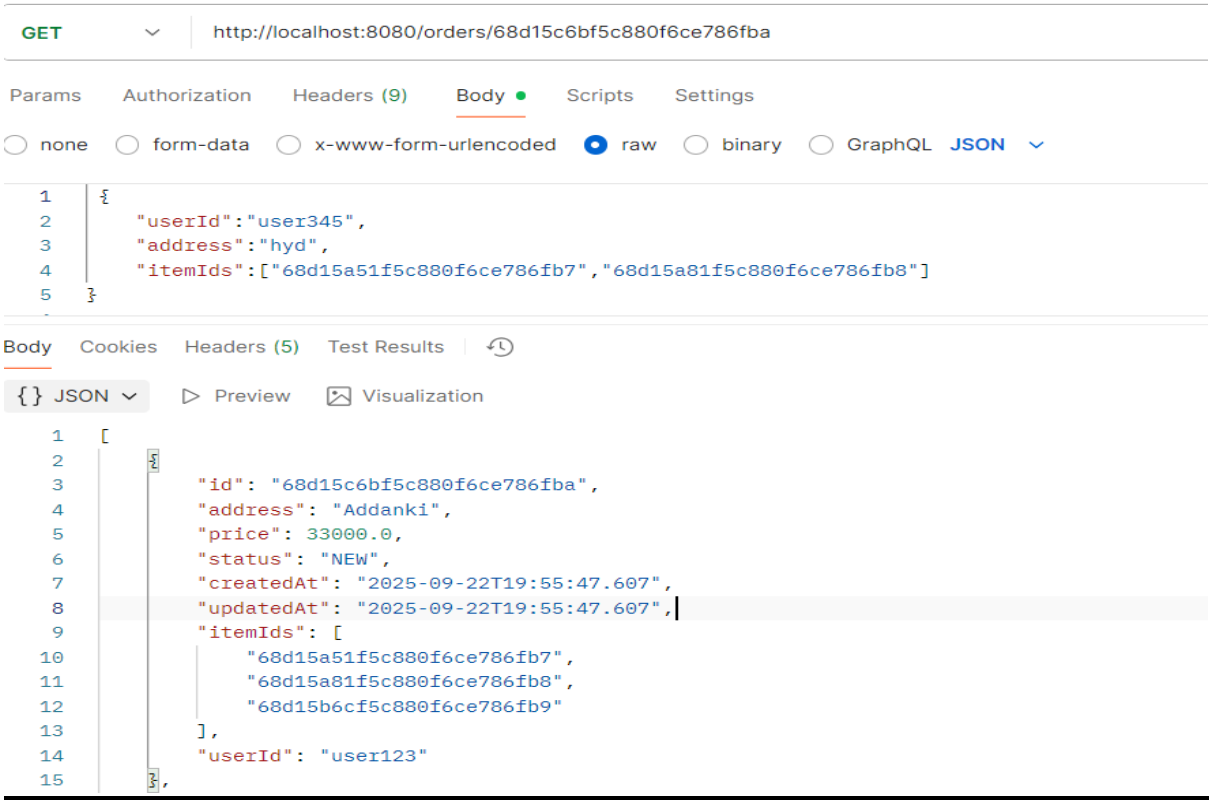
- **Method**: POST
- **URL**: http://localhost:8080/orders/adds





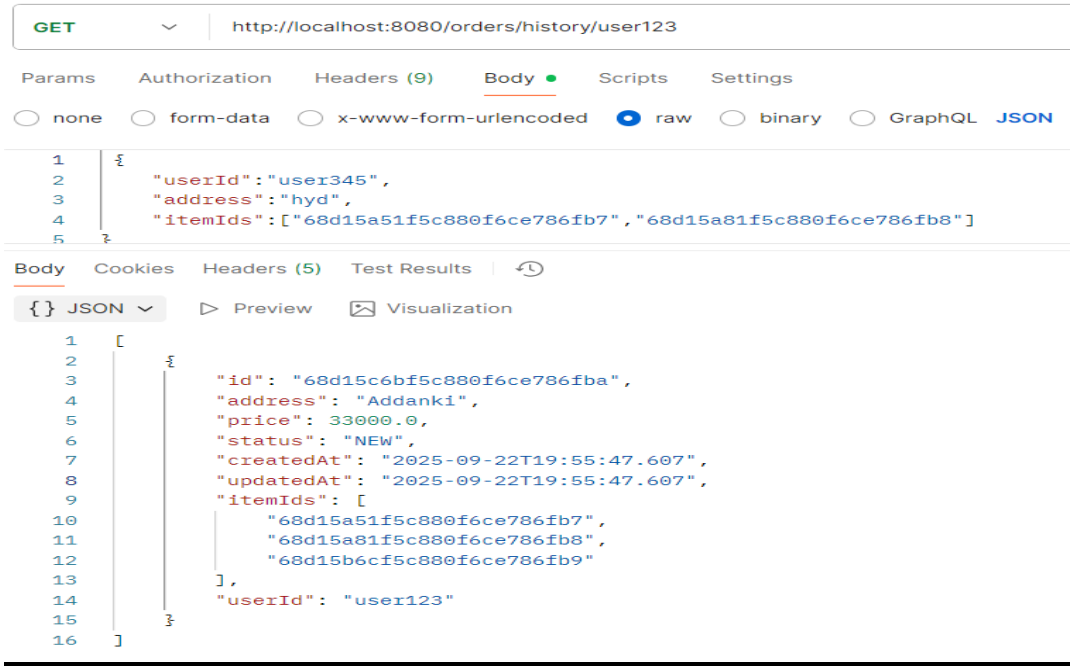
2.Get Order by Id

- **Method**: POST
- **URL**: http://localhost:8080/orders/{id}



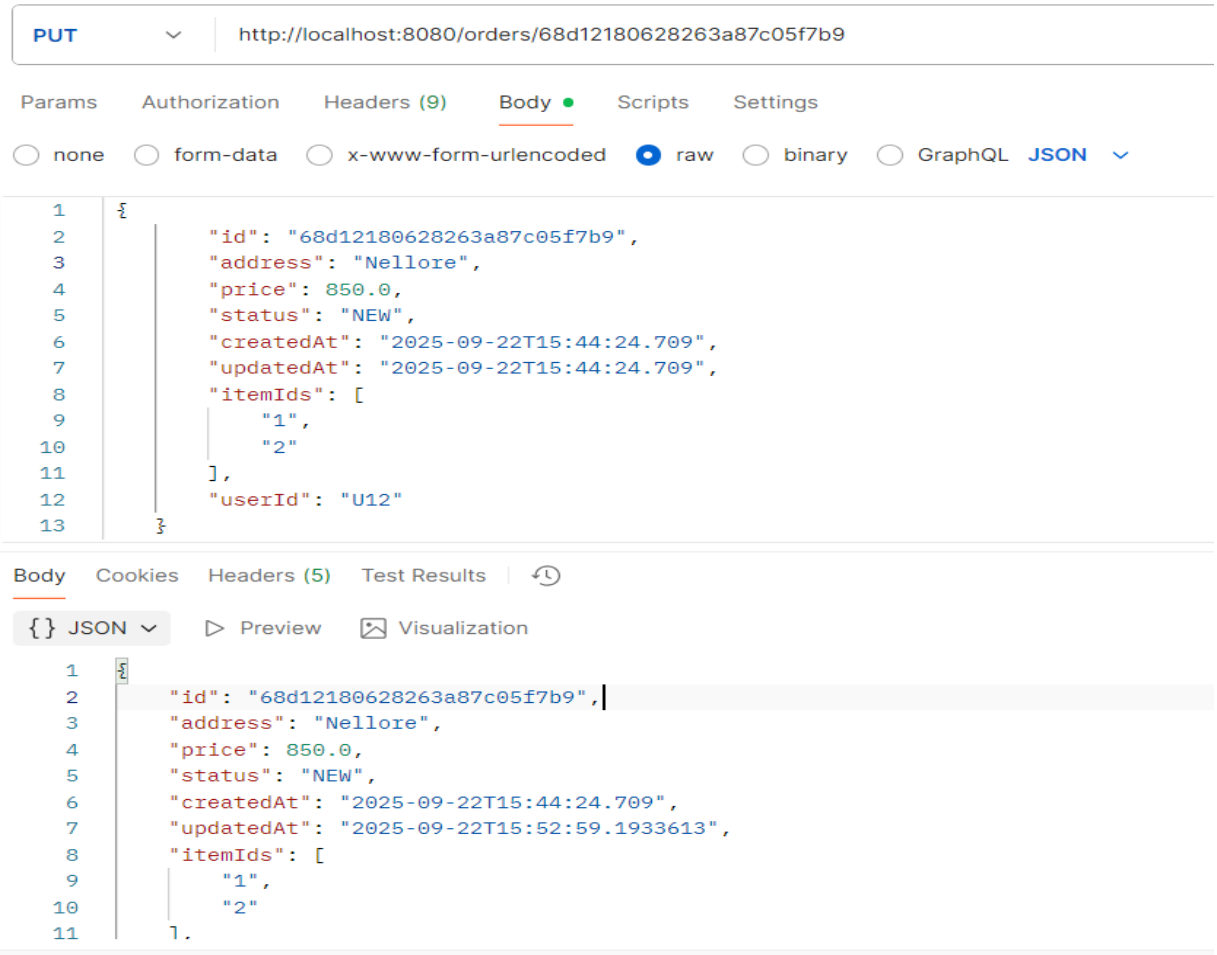
3.Get Order History by User

- **Method:** Get
- **URL:** http://localhost:8080/orders/history/{userId}



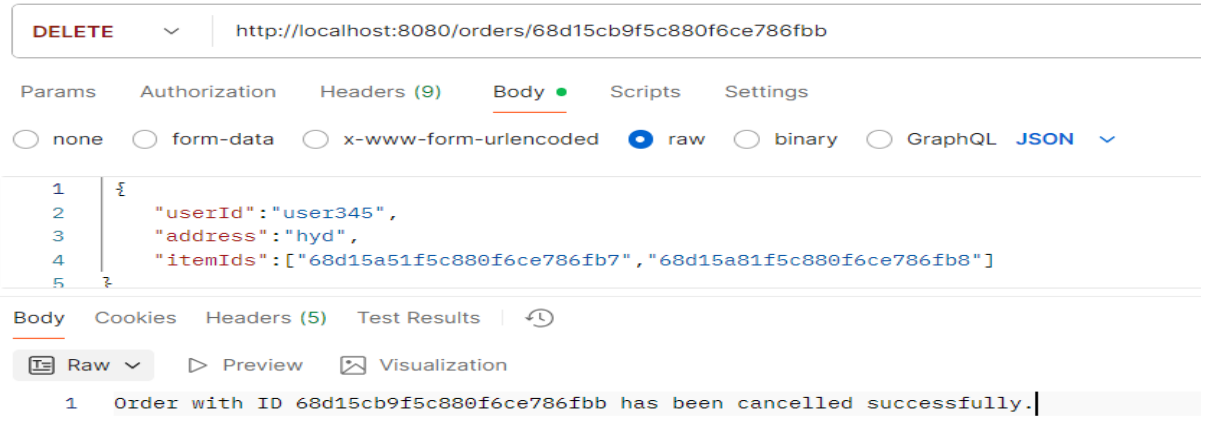
4.Update Order

- **Method:** put
- **URL:** http://localhost:8080/orders/{id}



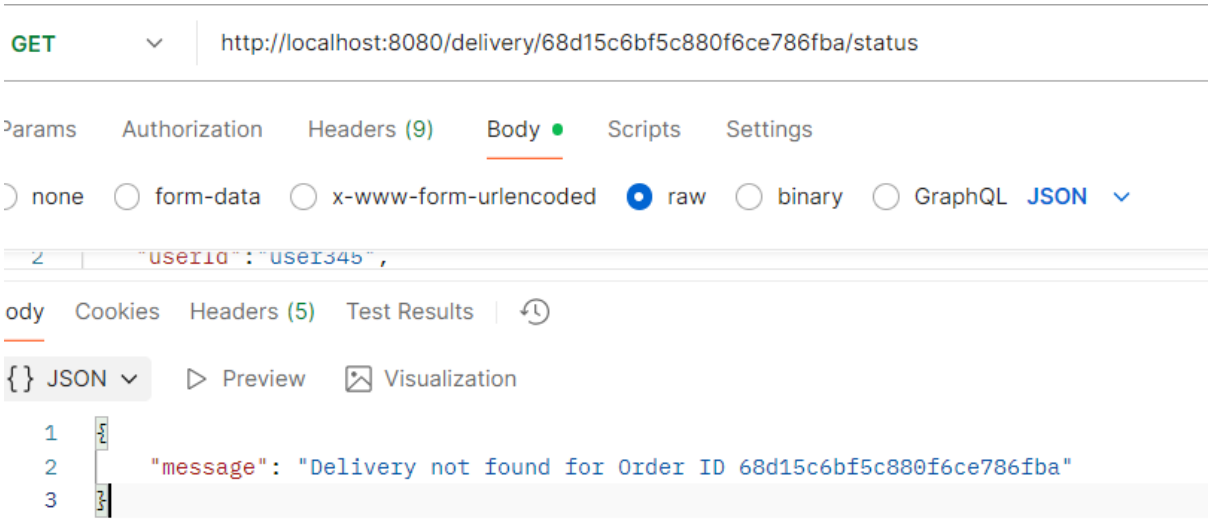
5.Delete Order

- **Method:** Delete
- **URL:** http://localhost:8080/orders/{id}



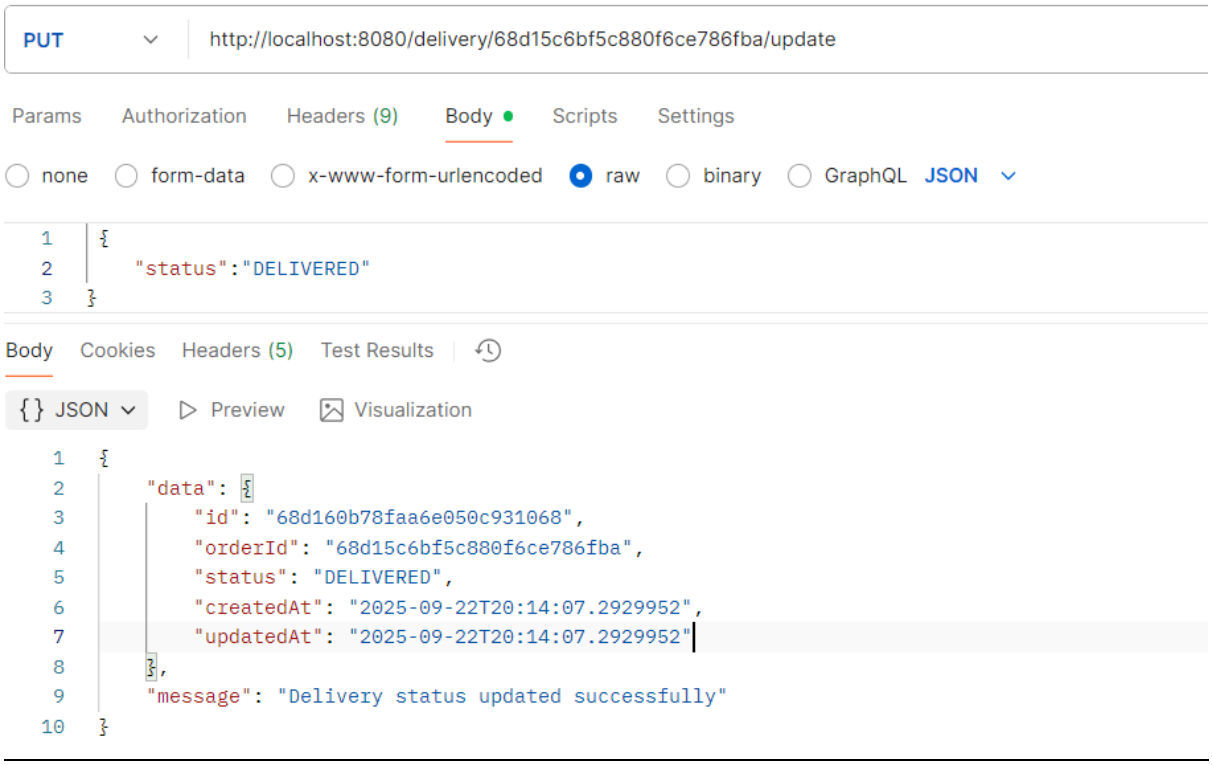
6.Get Delivery Status

- **Method:** GET
- **URL:** http://localhost:8080/delivery/{orderId}/status



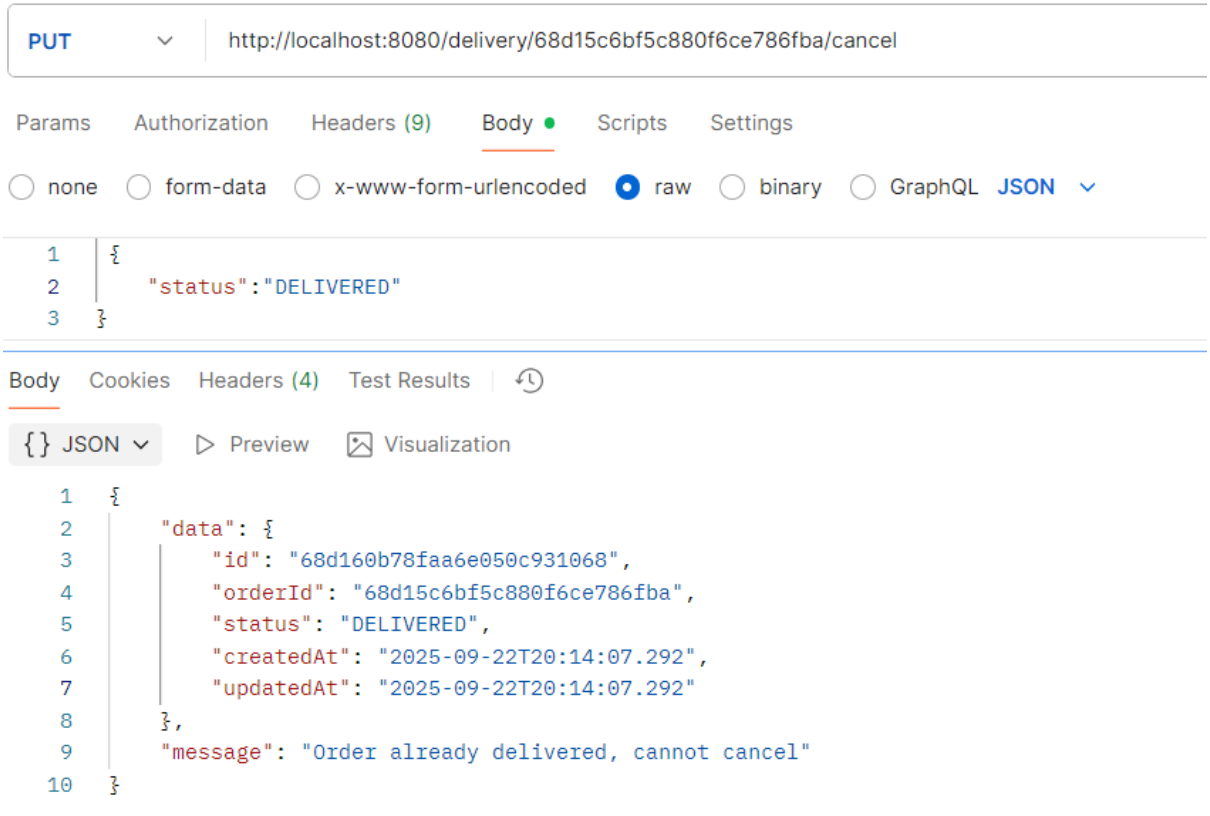
7.Update Delivery Status

- **Method:** PUT
- **URL:** http://localhost:8080/delivery/{orderId}/update



8.Cancel Order:

- **Method:**PUT
- **URL:** http://localhost:8080/delivery/{orderId}/cancel



9.Update Delivery Status

- **Method:** PUT
- **URL:** http://localhost:8080/delivery/{orderId}/update

10.Cancel Order:

- **Method:** PUT
- **URL:** http://localhost:8080/delivery/{orderId}/cancel