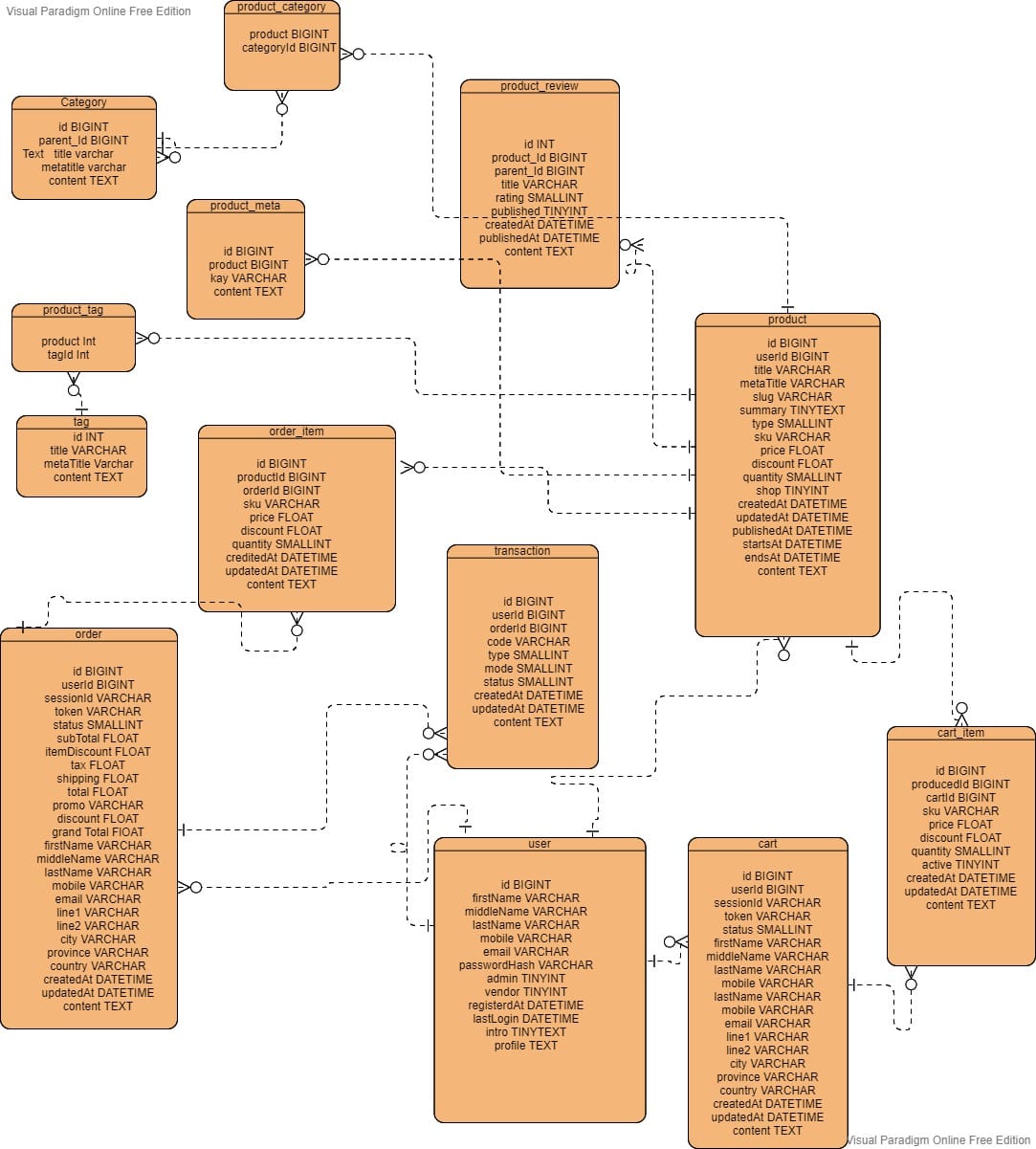
**ASSESSMENT(ECN-4138)**

**Answer(4):**

****

* **User Table:** User table stores user information. The same table can be used to manage different types of users including admins and customers.
* **Product Table:**Product Table to store the product data.It uses the column quantity to track the stock available in the product inventory to keep the design simple.
* **Product Meta:**The Product Meta Table can be used to store additional information about products including the product banner URL etc.
* **Product Review Table:**Product Review Table stores the product reviews.
* **Category Table and Product category Table:**Category Table and Product Category Table stores the product categories and their mappings
* **Cart Table and Cart item table:** The Cart Table and Cart Item Table can be made optional if the Local Data, Session, or in-memory database like Redis is used to store the cart data. The same can be referred to create the order on payment success.
* **Order Table and Order Item Table**:This section provides the tables to manage the store orders. A logged-in user can also be associated with the order.
* **Transaction Table:** transaction table track the order payments made by the buyer and for bookkeeping. We can also use the same table to record the partial or full refund of the order.
* **Address Table:**An address table can be used to avoid the redundant columns in the Cart and Order table depending on the actual implementation. It can be directly mapped to the Cart Table and Order Table using the appropriate foreign keys.

**Answer(5):**

**CRUD operation:**CRUD stands for "Create, Read, Update, and Delete," which are the four basic database operations. Many HTTP services also model CRUD operations through REST or REST-like APIs.

**1)Product Scenario:**

**package com.java.example.model;**

public class Product {

private Long id;

private String firstName;

private String lastName;

private String year;

// Default no-argument Constructor required

public Product() {}

public Product(String firstName, String lastName, String year) {

this.firstName = firstName;

this.lastName = lastName;

this.year = year;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getYear() {

return year;

}

public void setYear(String year) {

this.year = year;

}

}

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

**package com.java.example.repository;**

import java.util.Collection;

import java.util.HashMap;

import java.util.Map;

import java.util.Optional;

import org.springframework.stereotype.Repository;

import com.java.example.model.Product;

@Repository

public class ProductRepository {

Map<Long, Product> product= new HashMap<>();

long currentId = 100;

// Return all products

public Collection<Product> findAll(){

return products.values();

}

// Find the product with this id

public Optional<Product> findById(Long id) {

Product product = null;

if (products.containsKey(id)) product = products.get(id);

return Optional.ofNullable(product);

}

// Save a new product

public Product save(Product product) {

product.setId(++currentId);

product s.put(product.getId(), product);

return product;

}

// Update the product with this id

public Optional<Product> update(Product product) {

Product currentProduct = products.get(product.getId());

if (currentProduct != null) {

products.put(product.getId(), product);

currentProduct = products.get(product.getId());

}

return Optional.ofNullable(currentProduct);

}

// Delete product with this id

public Optional<Product> delete(Long id) {

Product currentProduct = products.get(id);

if (currentProduct != null) {

products.remove(id);

}

return Optional.ofNullable(currentProduct);

}

}

……………………………………………………………………………………………………………….

**package com.java.example.controller;**

import java.util.Collection;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

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.ResponseStatus;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

import com.java.example.model.Product;

import com.java.repository.ProductRepository;

@RestController

@RequestMapping("/products")

public class ProductController {

private final ProductRepository repository;

@Autowired

public ProductController(ProductRepository repository) {

this.repository = repository;

}

@SuppressWarnings("serial")

@ResponseStatus(HttpStatus.NOT\_FOUND)

class ProductNotFoundException extends RuntimeException {

public ProductNotFoundException() {

super("Productdoes not exist");

}

}

@GetMapping

Collection<Product> readproducts(){

return this.repository.findAll();

}

@GetMapping("/{id}")

Product readProduct(@PathVariable Long id) {

return this.repository.findById(id)

.orElseThrow(ProductNotFoundException::new);

}

@PostMapping

ResponseEntity<?> addProductt(@RequestBody Product product){

Product result = this.repository.save(product);

URI location = ServletUriComponentsBuilder

.fromCurrentRequest()

.path("/{id}")

.buildAndExpand(result.getId())

.toUri();

return ResponseEntity.created(location).build();

}

@PutMapping

Product updateProduct(@RequestBody Product product) {

return this.repository.update(product)

.orElseThrow(ProductNotFoundException::new);

}

@DeleteMapping("/{id}")

void deleteProduct(@PathVariable Long id) {

this.repository.delete(id)

.orElseThrow(ProductNotFoundException::new);

}

}

=====================================================================

**2)Order Scenario:**

**package com.java.example.model;**

public class Order {

private Long id;

private String firstName;

private String lastName;

private String year;

// Default no-argument Constructor required

public Order() {}

public Order(String firstName, String lastName, String year) {

this.firstName = firstName;

this.lastName = lastName;

this.year = year;

}

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getYear() {

return year;

}

public void setYear(String year) {

this.year = year;

}

}

==================================================================

**package com.java.example.repository;**

import java.util.Collection;

import java.util.HashMap;

import java.util.Map;

import java.util.Optional;

import org.springframework.stereotype.Repository;

import com.java.example.model.Order;

@Repository

public class Order Repository {

Map<Long, Order> orders = new HashMap<>();

long currentId = 100;

// Return all orders

public Collection<Order> findAll(){

return orders.values();

}

// Find the order with this id

public Optional<Order> findById(Long id) {

Order order = null;

if (orders.containsKey(id)) order = orders.get(id);

return Optional.ofNullable(order);

}

// Save a new order

public Order save(Order order) {

order.setId(++currentId);

orders.put(order.getId(), order);

return order;

}

// Update the student with this id

public Optional<Order> update(Order order) {

Order currentOrder = orders.get(order.getId());

if (currentOrder!= null) {

orders.put(order.getId(), order);

currentOrder = orders.get(order.getId());

}

return Optional.ofNullable(currentOrder);

}

// Delete order with this id

public Optional<Order> delete(Long id) {

Order currentOrder= orders.get(id);

if (currentOrder!= null) {

orders.remove(id);

}

return Optional.ofNullable(currentOrder);

}

}

===============================================

**package com.java.example.controller**;

import java.net.URI;

import java.util.Collection;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

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.ResponseStatus;

import org.springframework.web.bind.annotation.RestController;

import org.springframework.web.servlet.support.ServletUriComponentsBuilder;

import com.java.model.Order;

import com.java.example.repository.OrderRepository;

@RestController

@RequestMapping("/orders")

public class OrderController {

private final OrderRepository repository;

@Autowired

public OrderController(ProductRepository repository) {

this.repository = repository;

}

@SuppressWarnings("serial")

@ResponseStatus(HttpStatus.NOT\_FOUND)

class OrderNotFoundException extends RuntimeException {

public OrderNotFoundException() {

super("Order does not exist");

}

}

@GetMapping

Collection<Order> readorder(){

return this.repository.findAll();

}

@GetMapping("/{id}")

Order readOrder(@PathVariable Long id) {

return this.repository.findById(id)

.orElseThrow(OrderNotFoundException::new);

}

@PostMapping

ResponseEntity<?> addOrder(@RequestBody Order order){

Order result = this.repository.save(order);

URI location = ServletUriComponentsBuilder

.fromCurrentRequest()

.path("/{id}")

.buildAndExpand(result.getId())

.toUri();

return ResponseEntity.created(location).build();

}

@PutMapping

Order updateOrder(@RequestBody Order order) {

return this.repository.update(order)

.orElseThrow(OrderNotFoundException::new);

}

@DeleteMapping("/{id}")

void deleteOrder(@PathVariable Long id) {

this.repository.delete(id)

.orElseThrow(OrderNotFoundException::new);

}

}