**V2**

import ballerina/grpc;

import ballerina/uuid;

type Customer record{|

readonly string user\_id;

string user\_name;

string password;

string role;

|};

type Product record {|

readonly string product\_id;

string product\_name;

float product\_price;

int stock\_quantity;

int stock\_keeping\_unit;

|};

type Cart record {|

string cart\_id;

string user\_id;

string product\_id;

int quantity;

|};

table <Customer> customerTable = table{key(user\_id)};

table <Product>  productTable = table{key:(product\_id)};

table <Cart>  cartTable = table{key:(cart\_id)};

# Description.

@grpc:Descriptor {value: SHOPPING\_DESC}

service "Customer" on grpcListener {

        remote function AddCustomer(AddCustomerRequest value) returns AddCustomerResponse|error {

            Customer newCustomer = {user\_id:value.user\_id ,user\_name:value.user\_name,password:value.password,role:value.role};

        customerTable.add(newCustomer);

        AddCustomerResponse response = {user\_id: value.user\_id,role: value.role };

        return response;

    }

    remote function AuthenticateUser(AuthenticateCustomerRequest value) returns AuthenticateCustomerResponse|error {

        boolean isAuthenticated = false;

        foreach Customer customer in customerTable {

            if customer.user\_name == value.user\_name && customer.password == value.password{

                isAuthenticated = true;

                break;

            }

        }

        string message = isAuthenticated? "Authentication Successful" : "Authentication Failed";

        AuthenticateCustomerResponse response = {message : message};

        return response;

    }

    remote function ListAvailableProducts(Empty value) returns ListProductsResponse|error {

        ListProductsResponse response = {product: []};

        foreach Product product in productTable{

            response.products.push(product);

        }

            return response;

    }

    type SearchProductRequest record {

        string query

    };

    remote function SearchProduct(SearchProductRequest value) returns ProductDetailsResponse|error {

        ProductDetailsResponse response = {product: []};

        string searchQuery = value.query.toLowerAscii();

        foreach Product product in productTable{

            if product.product\_name.toLowerAscii().includes(searchQuery){

                response.product.push(product);

            }

        }

        return response;

    }

    remote function AddToCart(AddToCartRequest value) returns AddToCartResponse|error {

        string cart\_id = uuid:createTypeAsString();

        Cart newCartItem = {cart\_id:cart\_id,user\_id: value.user\_id,product\_id: value.product\_id,quantity: value.quantity};

        check cartTable.add(newCartItem);

        AddToCartResponse response = {cart\_id: cart\_id};

        return response;

     }

    remote function PlaceOrder(PlaceOrderRequest value) returns PlaceOrderResponse|error {

        string order\_id = uuid:createTypeAsString();

        PlaceOrderResponse response = {order\_id: order\_id};

        return response;

    }

    remote function CancelOrder(CancelOrderRequest value) return CancelOrderResponse|error{

        CancelOrderResponse response = {order\_id: value.order\_id, status: "Cancelled"};

        return response;

    }

}

function generateCartId() returns string {

    return "prod" + uuid:createTypeAsString();

}