

BAHRIA UNIVERSITY (KARACHI CAMPUS)

Open Ended Lab 2 – SPRING SEMESTER – 2023

(Software Design and Architecture - SEL457)

Class: **BSE 4A/B** (Morning)

Course Instructor: **Engr Majid Kaleem**

Lab Instructor: **Engr Muhammad Rehan Baig Max Marks: 6**

Student’s Name: AHSAN SAJJAD Reg. No: 79309

**Note:** Note: Probability of similarity is 0% copied and similar solutions will be marked as zero

**SCENERIO:**

StoreJinnie an online store after implementation of a system to manage their inventory, orders, and shipping processes in PHASEI. Now wanted to expand their application and also wants to add automation in their system. Multiple modules will be created using different tech stack (i.e.: Programming Languages) as separate projects because multiple technical teams working on modules to rapid application development.

Following are the main Modules of this system.

* **Payments Gateways**: Multiple Payment Gateways can be integrated into the system (Stripe, PayPal, Remitly, Easypaisa, etc.). if one payment gateway fails to process payments then the process automatically transfers to another payment gateway based on countries.
* **Notifications:** Based on user visits and purchases from the system Newsletter Emails, Text and In App Notifications will be scheduled and dispatched to Users.
* **Message Queueing:** The system can handle a high volume of orders without overwhelming the processing capacity, as the orders are queued and processed asynchronously.
* **Affiliate Program**: This Module will provide Affiliate related portals and marketing data to Affiliates.
* **Extensions:** This Module will provide Exposed Endpoints for Integration with other Applications and using of system public data to third party apps.

# Q1 Implement best Suited Architecture for this application

**Q2 Implement best Suited Architectural Patterns and Design Patterns for this Application.**

NOTE: Please solve the above scenario with explanations.

# AHSAN SAJJAD BSE4-B

# 02-131212-039

**EXPLAINATION:**

**Architectural Patterns:**

For StoreJinnie's system, a **microservices architecture** might be a good design choice. An architectural design known as microservices architecture organizes the application as a group of services. Each service has the option of having its own database and writing it in a separate programming language. This design enables quick application development, scalability, and simple integration with applications and services from third parties.

Each module in a microservices architecture can be created as a distinct service with its own technology stack and database. Additionally, this architecture makes it simple to scale individual modules according to their unique needs. The framework for developing, deploying, and autonomously maintaining microservices architectural diagrams and services is provided by microservices architecture.

**Design Patterns**

The Design patterns Which are suited to the given scenario are given below:

* **Observer Pattern:** Subscribers can be notified of data changes using the observer pattern. As a result, updating subscribers when data changes is made simpler.
* **Singleton Pattern:** This pattern can be used to guarantee that a class only has one instance. This is helpful for classes that must be available to everyone.
* **Factory Pattern:** Microservice instances can be created using the factory pattern. Microservice creation and management are now simpler as a result.
* **Repository Pattern:** A database's data can be accessed using the repository pattern. Data access and management are now simpler as a result.

**SOLUTION:**

**Implementation of Architecture Pattern and Design Pattern:**

**MAIN BODY:**

using oelfinal.AdapterPattern\_Extentions;

using oelfinal.AffiliateManagment;

using oelfinal.EventDriven\_Notification;

using oelfinal.factoryDesign\_Payment;

Console.WriteLine("------Welcome to StoreJinnie an online store-----");

Console.WriteLine("Select an option 1)Payment Gateway 2)Notifications and Messages 3)Affiliate 4)Extentions");

int ans=int.Parse(Console.ReadLine());

switch (ans)

{

case 1:

PaymentGatewayFactory gatewayFactory = new PaymentGatewayFactory();

Console.WriteLine("Select a payment gateway (stripe/paypal/Easypaisa):");

string gatewayType = Console.ReadLine();

try

{

PaymentGateway selectedGateway = gatewayFactory.CreatePaymentGateway(gatewayType);

Console.WriteLine("Enter the payment amount:");

double amount = double.Parse(Console.ReadLine());

selectedGateway.ProcessPayment(amount);

}

catch (ArgumentException ex)

{

Console.WriteLine(ex.Message);

}

break;

case 2:

IEventBus eventBus = new RabbitMQEventBus();

NotificationMicroservice notificationsMicroservice = new NotificationMicroservice(eventBus);

notificationsMicroservice.StartListening();

UserVisitEvent visitEvent = new UserVisitEvent { UserId = "123" };

UserPurchaseEvent purchaseEvent = new UserPurchaseEvent { UserId = "456" };

eventBus.Publish(visitEvent);

eventBus.Publish(purchaseEvent);

break;

case 3:

AffiliateManagmentService affiliateManagementService = new AffiliateManagmentService();

AffiliatePortalService affiliatePortalService = new AffiliatePortalService(affiliateManagementService);

affiliatePortalService.RegisterNewAffiliate("StoreJinnie");

List<Affiliate> affiliates = affiliatePortalService.GetAllAffiliates();

foreach (var affiliate in affiliates)

{

Console.WriteLine($"Affiliate ID: {affiliate.Id}, Name: {affiliate.Name}");

}

break;

case 4:

PublicDataSystem publicDataSystem = new PublicDataSystem();

IIntegrationAdapter integrationAdapter = new PublicDataSystemAdapter(publicDataSystem);

var publicData = integrationAdapter.GetPublicData();

Console.WriteLine("Retrieved Public Data:");

Console.WriteLine(publicData.ToString());

Console.WriteLine("Press any key to exit...");

Console.ReadKey();

break;

default:

Console.WriteLine("Select Valid Option");

break;

**Payment Gateway Class**

public abstract class PaymentGateway

{

public abstract void ProcessPayment(double amount);

}

public class EasypaisaPaymentGateway:PaymentGateway

{

public override void ProcessPayment(double amount)

{

//code of easypaisa

}

}

public class PayPalPaymentGateway:PaymentGateway

{

public override void ProcessPayment(double amount)

{

//code of paypal

}

}

public class StripePaymentGateway:PaymentGateway

{

public override void ProcessPayment(double amount)

{

//code of stripe

}

}

public PaymentGateway CreatePaymentGateway(string gatewayType)

{

PaymentGateway paymentGateway = null;

switch (gatewayType.ToLower())

{

case "stripe":

paymentGateway = new StripePaymentGateway();

break;

case "paypal":

paymentGateway = new PayPalPaymentGateway();

break;

case "easypaisa":

paymentGateway = new EasypaisaPaymentGateway();

break;

default:

throw new ArgumentException("Invalid gateway type.");

}

return paymentGateway;

}

**Microservices Architecture:**

public interface IEventBus

{

void Publish<TEvent>(TEvent @event);

void Subscribe<TEvent, TEventHandler>() where TEventHandler : IEventHandler<TEvent>;

}

public interface IEventHandler<TEvent>

{

void Handle(TEvent @event);

}

public class NotificationMicroservice

{

private readonly IEventBus \_eventBus;

public NotificationMicroservice(IEventBus eventBus)

{

\_eventBus = eventBus;

}

public void StartListening()

{

\_eventBus.Subscribe<UserVisitEvent, UserVisitEventHandler>();

\_eventBus.Subscribe<UserPurchaseEvent, UserPurchaseEventHandler>();

}

}

public class RabbitMQEventBus : IEventBus

{

public void Publish<TEvent>(TEvent @event)

{

}

public void Subscribe<TEvent, TEventHandler>() where TEventHandler : IEventHandler<TEvent>

{

}

}

public class UserPurchaseEvent

{

public string UserId { get; set; }

}

public class UserPurchaseEventHandler : IEventHandler<UserPurchaseEvent>

{

public void Handle(UserPurchaseEvent @event)

{

Console.WriteLine($"Processing user purchase event for UserId: {@event.UserId}");

}

}

public class UserVisitEvent

{

public string UserId { get; set; }

}

public class UserVisitEventHandler : IEventHandler<UserVisitEvent>

{

public void Handle(UserVisitEvent @event)

{

Console.WriteLine($"Processing user visit event for UserId: {@event.UserId}");

}

}

**Affiliate Program:**

public class Affiliate

{

public int Id { get; set; }

public string Name { get; set; }

}

public class AffiliateManagmentService

{

private List<Affiliate> \_affiliates;

public AffiliateManagmentService()

{

\_affiliates = new List<Affiliate>();

}

public void RegisterAffiliate(string name)

{

int newId = \_affiliates.Count + 1;

var newAffiliate = new Affiliate { Id = newId, Name = name };

\_affiliates.Add(newAffiliate);

}

public List<Affiliate> GetAllAffiliates()

{

return \_affiliates;

}

public class AffiliatePortalService

{

private readonly AffiliateManagmentService \_affiliateManagementService;

public AffiliatePortalService(AffiliateManagmentService affiliateManagementService)

{

\_affiliateManagementService = affiliateManagementService;

}

public void RegisterNewAffiliate(string name)

{

\_affiliateManagementService.RegisterAffiliate(name);

}

public List<Affiliate> GetAllAffiliates()

{

return \_affiliateManagementService.GetAllAffiliates();

}

}

}

**Extensions:**

public interface IIntegrationAdapter

{

object GetPublicData();

}

public class PublicDataSystem

{

public object GetPublicData()

{

return new object();

}

}

public class PublicDataSystemAdapter : IIntegrationAdapter

{

private readonly PublicDataSystem \_publicDataSystem;

public PublicDataSystemAdapter(PublicDataSystem publicDataSystem)

{

\_publicDataSystem = publicDataSystem;

}

public object GetPublicData()

{

return \_publicDataSystem.GetPublicData();

}

}