**1. Overview**

The Pixelz Checkout System is a microservice-based architecture designed to handle e-commerce order processing, including:

Order Management: Get All, Searching, and checking out orders.

Payment Processing: Handling simulated payment requests.

Production Submission: Submitting successful orders to an internal production system.

Email Notifications: Sending confirmation emails after successful checkout.

The system is modular, scalable, and uses REST APIs for communication between services. Each service is responsible for a single business capability (Single Responsibility Principle).

**2. Core Components**

**2.1 OrderService**

Exposes REST APIs for:

* Get all orders
* Searching orders by name
* Checkout operation

Uses In-Memory Database for storing orders

Communicates with:

* PaymentService (to process payment)
* ProductionService (to push order for processing)
* EmailService (to send email confirmation)

Swagger enabled for API documentation

**2.2 PaymentService**

Mocks a payment gateway

Accepts a **POST /api/payment/process** request with OrderId and Amount

Returns success or failure randomly or based on rules (e.g., TotalAmount % 2 != 0 → fail)

**2.3 ProductionService**

Simulates sending orders to the internal production system

Endpoint: POST /api/production/submit

Returns success if request meets mock conditions

**2.4 EmailService**

Simulates sending email notifications

Endpoint: POST /api/email/send

Always returns success (configurable for testing failure scenarios)

**3. Data Models**

Order

public class Order

{

public Guid Id { get; set; }

public string Name { get; set; } = null!;

public Guid CustomerId { get; set; }

public OrderStatus Status { get; set; } = OrderStatus.Pending;

public DateTime CreatedAt { get; set; }

public DateTime? PaidAt { get; set; }

public decimal TotalAmount { get; set; }

public Customer Customer { get; set; }

public List<OrderItem> Items { get; set; } = new();

}

OrderItem

public class OrderItem

{

public Guid Id { get; set; }

public Guid OrderId { get; set; }

public Guid ProductId { get; set; }

public string ProductName { get; set; } = null!;

public int Quantity { get; set; }

public decimal UnitPrice { get; set; }

public decimal Total => Quantity \* UnitPrice;

public Order? Order { get; set; }

}

Customer

public class Customer

{

public Guid Id { get; set; }

public string FullName { get; set; } = null!;

public string Email { get; set; } = null!;

public string Phone { get; set; } = null!;

}

Enum: OrderStatus

public enum OrderStatus

{

Pending = 0,

PaymentFailed = 1,

PaymentSuccessful = 2,

ProductionFailed = 3,

Completed = 4

}

**4. Key APIs**

OrderService

GET /api/order/orders => Get all order ( used to find the customerId and orderId that pre-generate by system)

GET /api/order/{customerId}/orders?customerid={customerid}&name={orderName} → Search orders by name

POST /api/order/{customerId}/checkout/{orderId}/ → Checkout an order

PaymentService

POST /api/payment/process → Process payment for an order

ProductionService

POST /api/production/submit → Submit order to production

EmailService

POST /api/email/send → Send confirmation email

All APIs return ApiResponse<T> with:

Result → Data or boolean

Status → Success | Fail | Error

Message → Description

6. Component Interaction (Checkout Flow)

Art Director calls:

POST /api/order/{customerId}/checkout/{orderId}/

OrderService:

Validates order exists and is Pending

Calls PaymentService

If fail → Update OrderStatus = PaymentFailed → Stop

Calls ProductionService

If fail → Update OrderStatus = ProductionFailed → Stop

Calls EmailService

Send confirmation email

Updates order status to Completed

7. Configuration

Service URLs are stored in appsettings.json:

"Services": {

"Payment": {

"BaseUrl": "https://localhost:7109/",

"Endpoints": {

"Process": "api/payment/process"

}

},

"Production": {

"BaseUrl": "https://localhost:7250/",

"Endpoints": {

"Submit": "api/production/submit"

}

},

"Email": {

"BaseUrl": "https://localhost:7030/",

"Endpoints": {

"Send": "api/email/send"

}

}

},

HttpClientFactory registers clients with these base URLs for external calls.

9. Non-Functional Considerations

Scalability: Each service can be scaled independently

Fault Tolerance: No retry or circuit breaker implemented (future scope)

Security: Authentication/Authorization is not implemented in this version

Performance: In-memory DB for simplicity; persistent DB for production