## **Задание**

Провести код-ревью разработанного микросервиса.

## **Условие**

Разработчику была поставлена задача реализовать микросервис заказов для интернет-магазина в виде API. Микросервис должен содержать три API метода: для создания заказа, завершения заказа и получения информации и последних заказах.

### **Описание методов:**

1. **Создание новых заказов** Сигнатура:

| {  "sum": 1000, //общая сумма заказа  "contractorType": 1, // тип контрагента (юридическое/физическое лицо)  "items": [  {  "productId": 1, // ID товара  "price": 1000, // стоимость товара  "quantity": 1 // количество  },  //...  ] } |
| --- |

* При создании заказа генерируется уникальный номер в формате {год|2020}-{месяц|09}-{порядковый номер заказа}. Например 2020-09-12345
* После создания заказа физических лиц должно редиректить на страницу оплаты http://some-pay-agregator.com/pay/{номер заказа}

1. **Завершение заказа с проверкой оплаты**   
   Метод осуществляет проверку факта оплаты для показа пользователю и по результату пользователь должен увидеть либо страницу благодарности, либо напоминание о необходимости оплаты.

*Для юридических лиц* - проверка факта оплаты через отдельный микро-сервис (реализация обращения к микросервису на код ревью не представлена)

*Для физических* - проверка через флаг оплаты, поле в БД (данные попадают в базу через внешний микросервис, реализация обмена данными на код ревью не представлена).

1. **Получения информации о N последних заказах**Метод должен возвращать информацию о заданном количестве заказов в формате:

| [  {  "id": "2020-09-123456", // номер заказа  "sum": 1000, //общая сумма заказа  "contractorType": 1, // тип контрагента (юридическое/физическое лицо)  "items": [  {  "productId": 1, // ID товара  "price": 1000, // стоимость товара  "quantity": 1 // количество  },  //...  ]  },  // ... ] |
| --- |

## 

## Реализация

#### Содержимое файла migrations/init.sql

| CREATE TABLE orders (  id VARCHAR(20) NOT NULL PRIMARY KEY,  sum INT DEFAULT 0,  contractor\_type SMALLINT,  is\_paid SMALLINT DEFAULT 0,  createdAt TIMESTAMP DEFAULT NOW() ) CHARACTER SET utf8 COLLATE utf8\_general\_ci engine MyISAM;  CREATE TABLE order\_products (  id INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,  order\_id VARCHAR(20),  product\_id INT,  price INT DEFAULT 0,  quantity INT DEFAULT 1,  FOREIGN KEY (order\_id) REFERENCES orders(id) ON DELETE SET NULL ) CHARACTER SET utf8 COLLATE utf8\_general\_ci engine MyISAM; |
| --- |

#### Содержимое файла src\Entity\Order.php

| <?php  namespace App\Entity;  use App\Service\BillGenerator; use App\Service\BillMicroserviceClient;  const CONTRACTOR\_TYPE\_PERSON = 1; const CONTRACTOR\_TYPE\_LEGAL = 2;  class Order {  */\*\* @var string \*/*  public $id;   */\*\* @var int \*/*  public $sum;   */\*\* @var Item[] \*/*  public $items = [];   */\*\* @var int \*/*  public $contractorType;   */\*\* @var bool \*/*  public $isPaid;   */\*\* @var BillGenerator \*/*  public $billGenerator;   */\*\* @var BillMicroserviceClient \*/*  public $billMicroserviceClient;   */\*\*  \* @param string $id  \*/*  public function \_\_construct($id)  {  $this->id = $id;  }   public function getPayUrl()  {  return "http://some-pay-agregator.com/pay/" . $this->id;  }   public function setBillGenerator($billGenerator)  {  $this->billGenerator = $billGenerator;  }   public function getBillUrl()  {  return $this->billGenerator->generate($this);  }   public function setBillClient(BillMicroserviceClient $cl)  {  $this->billMicroserviceClient = $cl;  }   public function isPaid()  {  if ($this->contractorType == CONTRACTOR\_TYPE\_PERSON) {  return $this->isPaid;  }  if ($this->contractorType == CONTRACTOR\_TYPE\_LEGAL) {  return $this->billMicroserviceClient->IsPaid($this->id);  }  } } |
| --- |

#### Содержимое файла src\Entity\Item.php

| <?php  namespace App\Entity;  class Item {  */\*\* @var string \*/*  protected $id;   */\*\* @var string \*/*  protected $orderId;   */\*\* @var string \*/*  protected $productId;   */\*\* @var string \*/*  protected $price;   */\*\* @var string \*/*  protected $quantity;   */\*\*  \* @param string $orderId  \* @param string $productId  \* @param string $price  \* @param string $quantity  \*/*  public function \_\_construct($orderId, $productId, $price, $quantity)  {  $this->orderId = $orderId;  $this->productId = $productId;  $this->price = $price;  $this->quantity = $quantity;  }   */\*\*  \* @param int $id  \*/*  public function setId(int $id)  {  $this->id = $id;  }   */\*\*  \* @return int  \*/*  public function getId()  {  return $this->id;  }   */\*\*  \* @return int  \*/*  public function getOrderId()  {  return $this->orderId;  }   */\*\*  \* @param int $orderId  \*/*  public function setOrderId(int $orderId)  {  $this->orderId = $orderId;  }   */\*\*  \* @return int  \*/*  public function getProductId(): int  {  return $this->productId;  }   */\*\*  \* @param int $productId  \* @return Item  \*/*  public function setProductId(int $productId): self  {  $this->productId = $productId;  return $this;  }   */\*\*  \* @return int  \*/*  public function getPrice()  {  return $this->price;  }   */\*\*  \* @param int $price  \*/*  public function setPrice(int $price)  {  $this->price = $price;  }   */\*\*  \* @return int  \*/*  public function getQuantity()  {  return $this->quantity;  }   */\*\*  \* @param int $quantity  \*/*  public function setQuantity(int $quantity)  {  $this->quantity = $quantity;  } } |
| --- |

#### Содержимое файла src\Controller\OrderController.php

| <?php  namespace App\Controller;  use App\Factory\OrderFactory; use App\Repository\OrderRepository; use App\Service\BillGenerator; use App\Service\BillMicroserviceClient; use Symfony\Component\HttpFoundation\JsonResponse; use Symfony\Component\HttpFoundation\RedirectResponse; use \Symfony\Component\HttpFoundation\Request; use Symfony\Component\HttpFoundation\Response; use Symfony\Component\Routing\Annotation\Route; use const App\Entity\CONTRACTOR\_TYPE\_LEGAL; use const App\Entity\CONTRACTOR\_TYPE\_PERSON;  class OrderController {  */\*\* @var OrderFactory \*/*  protected $order\_factory;   */\*\* @var OrderRepository \*/*  protected $order\_repository;   public function \_\_construct(OrderFactory $order\_factory, OrderRepository $order\_repository)  {  $this->order\_factory = $order\_factory;  $this->order\_repository = $order\_repository;  }   */\*\*  \* @Route("/create", methods={"POST"})  \*/*  public function create(Request $request)  {  $orderData = json\_decode($request->getContent(), true);  $orderId = $this->order\_factory->generateOrderId();   try {  $order = $this->order\_factory->createOrder($orderData, $orderId);   if ($order->contractorType === CONTRACTOR\_TYPE\_PERSON) {  $this->order\_repository->save($order);  return new RedirectResponse($order->getPayUrl());   }  if ($order->contractorType === CONTRACTOR\_TYPE\_LEGAL) {  $order->setBillGenerator(new BillGenerator());  $this->order\_repository->save($order);  return new RedirectResponse($order->getBillUrl());  }  } catch (\Exception $exception) {  return new Response("Something went wrong");  }  }   */\*\*  \* @Route("/finish/{orderId}", methods={"GET"})  \*/*  public function finish($orderId)  {  $order = $this->order\_repository->get($orderId);  if ($order->contractorType == CONTRACTOR\_TYPE\_LEGAL) {  $order->setBillClient(new BillMicroserviceClient());  }  if ($order->isPaid()) {  return new Response("Thank you");  } else {  return new Response("You haven't paid bill yet");  }  }   */\*\*  \* @Route("/last", methods={"GET"})  \*/*  public function last(Request $request)  {  $limit = $request->get("limit");  $orders = $this->order\_repository->last($limit);  return new JsonResponse($orders);  } } |
| --- |

#### Содержимое файла src\Factory\OrderFactory.php

| <?php  namespace App\Factory;  use App\Entity\Item; use App\Entity\Order;  class OrderFactory {  */\*\* @var \PDO \*/*  protected $pdo;   */\*\*  \* OrderFactory constructor.  \* @param \PDO $pdo  \*/*  public function \_\_construct(\PDO $pdo)  {  $this->pdo = $pdo;  }   public function generateOrderId()  {  $sql = "SELECT id FROM order1s ORDER BY createdAt DESC LIMIT 1";  $result = $this->pdo->query($sql)->fetch();  return (new \DateTime())->format("Y-m") . "-" . $result['id'] + 1;  }   public function createOrder($data, $id)  {  $order = new Order($id);  foreach ($data as $key => $value)  {  if ($key == 'items')  {  foreach ($value as $itemValue) {  $order->items[] = new Item($id, $itemValue['productId'], $itemValue['price'], $itemValue['quantity']);  }  continue;  }  $order->{$key} = $value;  }  return $order;  } } |
| --- |

#### Содержимое файла src\Repository\OrderRepository.php

| <?php  namespace App\Repository;  use App\Entity\Item; use App\Entity\Order;  class OrderRepository {  */\*\* @var \PDO \*/*  protected $pdo;   */\*\*  \* @param \PDO $pdo  \*/*  public function \_\_construct(\PDO $pdo)  {  $this->pdo = $pdo;  }   public function save(Order $order)  {  $sql = "INSERT INTO orders (id, sum, contractor\_type) VALUES ({$order->id}, {$order->sum}, {$order->contractorType})";  $stmt = $this->pdo->prepare($sql);  $stmt->execute();  foreach ($order->items as $item) {  $sql = "INSERT   INTO order\_products (order\_id,product\_id,price,quantity)   VALUES ({$order->id},{$item->getProductId()},{$item->getPrice()}, {$item->getQuantity()})";  $stmt = $this->pdo->prepare($sql);  $stmt->execute();  }  }   */\*\* @return Order \*/*  public function get($orderId)  {  $sql = "SELECT \* FROM orders WHERE id={$orderId} LIMIT 1";  $stmt = $this->pdo->prepare($sql);  $data = $stmt->fetch();   $order = new Order($data['id']);  $order->contractorType = $data['contractor\_type'];  $order->isPaid = $data['is\_paid'];  $order->sum = $data['sum'];  $order->items = $this->getOrderItems($data['id']);   return $order;  }   */\*\* @return Order[] \*/*  public function last($limit = 10)  {  $sql = "SELECT \* FROM orders ORDER BY createdAt DESC LIMIT {$limit}";  $stmt = $this->pdo->prepare($sql);  $stmt->execute();  $data = $stmt->fetchAll();  $orders = [];  foreach ($data as $item) {  $order = new Order($item['id']);  $order->contractorType = $item['contractor\_type'];  $order->isPaid = $item['is\_paid'];  $order->sum = $item['sum'];  $order->items = $this->getOrderItems($item['id']);  $orders[] = $order;  }  return $orders;  }   public function getOrderItems($orderId)  {  $sql = "SELECT \* FROM order\_products WHERE order\_id={$orderId}";  $stmt = $this->pdo->prepare($sql);  $stmt->execute();  $data = $stmt->fetchAll();   $items = [];  foreach ($data as $item) {  $items[] = new Item($item['order\_id'], $item['product\_id'], $item['price'], $item['quantity']);  }  return $items;  } } |
| --- |