Модификация списка товаров

Есть платформа, предоставляющая заказчику возможности маркетплейса для розничной реализации своих продуктов. Платформа работает в продакшне и имеет определенное количество активных пользователей. Пользователи взаимодействуют с системой с помощью набора web и мобильных приложений для разных операционных систем. Взаимодействие клиентских приложений с бэкендом осуществляется через RESTful API.

Необходимо реализовать доработки каталога товаров для бэкенда и API системы на основе требований заказчика.

Функциональные требования:

- Избранное
  - предоставить авторизованному пользователю возможность добавлять продукты в избранное
  - избранные продукты должны отображаться с соответствующей пометкой в общем списке всех продуктов
  - пользователь так же должен иметь возможность просматривать список избранных продуктов отдельно
  - в списке избранных продуктов должны сохраниться фильтры и сортировка как и в общем списке продуктов
  - гостю функционал избранного недоступен, но он по прежнему должен иметь возможность просматривать список продуктов
- Детали продукта
  - необходимо добавить возможность отображать для продукта более одного изображения
  - существующие изображения должны остаться в текущем хранилище
  - для новых изображений принято решение использовать сервис на основе AWS S3

Остальные требования заказчика:

- минимизировать необходимые доработки, в том числе со стороны клиентских приложений
- сохранить обратную совместимость API, так как разные команды клиентских решений планирует релизы обновлений в разное время
- обеспечить стабильность и соответствие решения поставленной задаче с учетом будущих изменений системы

A. Документация API Для команд разработчиков клиентских приложений необходимо предоставить обновленную документацию RESTful API системы. Ниже приведены выдержка из существующей

документации о работе со списками продуктов. Опишите вариант изменения API с учетом всех требований заказчика и best практик проектирования RESTful API. Так же необходимо учесть и описать возможные ошибки.

Request example

```
GET /products?category=category-1&sort=name HTTP/1.1
Host: api.market.com
Authorization: Bearer 2YotnFZFEjr1zCsicMWpAA
Accept: application/json
```

Response example

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8

[
    {
        "id": 1,
        "name": "Example product 1",
        "description": "Example product 1 description",
        "image_url": "https://cdn.market.com/images/products/product_1.png",
        "category": "category-1"
    },
    {
        "id": 4,
        "name": "Example product 4",
        "description": "Example product 4 description",
        "image_url": "https://cdn.market.com/images/products/product_4.png",
        "category": "category-1"
    },
    ...
]
```

B. Реализация Реализация класса `Market\Product`, используемая для представления объекта в ответе API приведена ниже.

```php
<?php

namespace Market;

/**
 * Represents a single product record stored in DB.
 */
class Product
{
    /*...*/
```

```php
/**
 * @var FileStorageRepository
 */
private FileStorageRepository $storage;

/**
 * @var string
 */
private string $imageFileName;


/**
 * @param FileStorageRepository $fileStorageRepository
 */
public function __construct(FileStorageRepository $fileStorageRepository)
{
    $this->storage = $fileStorageRepository;
}

/*...*/

/**
 * Returns product image URL.
 *
 * @return string|null
 */
public function getImageUrl(): ?string
{
    if ($this->storage->fileExists($this->imageFileName) !== true) {
        return null;
    }

    return $this->storage->getUrl($this->imageFileName);
}

/**
 * Returns whether image was successfully updated or not.
 *
 * @return bool
 */
public function updateImage(): bool
{
    /*...*/

    try {
```

```
            if ($this->storage->fileExists($this->imageFileName) !== true) {
                $this->storage->deleteFile($this->imageFileName);
            }

            $this->storage->saveFile($this->imageFileName);
        } catch (\Exception $exception) {
            /*...*/

            return false;
        }

        /*...*/

        return true;
    }

    /*...*/
}
```

Класс `Market\FileStorageRepository` используется для работы с файловой системой и получения ссылки на изображения в статическом хранилище.

```
<?php

namespace Market;

/**
 * Repository for Market's filesystem and static storage.
 */
final class FileStorageRepository
{
    /**
     * Returns image URL or null.
     *
     * @param $fileName
     * @return string|null
     */
    public function getUrl($fileName): ?string
    {
        /*...*/
    }

    /**
     * Returns whether file exists or not.
     *
     * @param string $fileName
```

```
     * @return bool
     */
    public function fileExists(string $fileName): bool
    {
        /*...*/
    }

    /**
     * Deletes a file in the filesystem and throws an exception in case of errors.
     *
     * @param string $fileName
     * @return void
     */
    public function deleteFile(string $fileName): void
    {
        /*...*/
    }

    /**
     * Saves a file in the filesystem and throws an exception in case of errors.
     *
     * @param string $fileName
     * @return void
     */
    public function saveFile(string $fileName): void
    {
        /*...*/
    }
}
```

Было принято решение интегрировать стороннюю библиотеку для работы с сервисами AWS. Ниже приведены основные интерфейсы библиотеки, необходимые для реализации.

```
<?php

namespace AwsS3\Client;

use AwsS3\AwsUrlInterface;
use Exception;

/**
 *
 */
interface AwsStorageInterface
{
    /*...*/
```

```
    /**
     * Returns whether S3 connection is authorized or not.
     *
     * @return bool
     */
    public function isAuthorized(): bool;

    /*...*/

    /**
     * Returns AwsUrlInterface instance and throws an exception in case
     * connection or authorization errors.
     *
     * @param string $fileName
     * @return AwsUrlInterface
     * @throws Exception
     */
    public function getUrl(string $fileName): AwsUrlInterface;

    /*...*/
}
```

```
<?php

namespace AwsS3;

/**
 *
 */
interface AwsUrlInterface
{
    /*...*/

    /**
     * Returns string representation of the instance URL.
     *
     * @return string
     */
    public function __toString(): string;

    /*...*/
}
```

Необходимо продемонстрировать реализацию решения получения набора картинок для продукта. Допустим рефакторинг, внедрение новых сущностей и структурных единиц. Предполагается

что в проекте уже подключены необходимые компоненты, включая контейнер внедрения зависимостей.

- при реализации необходимо придерживаться принципов **SOLID и DRY**
- изменения допустимы только в неймспейсе `Market`, так как библиотека `AwsS3` поддерживается сторонней командой разработчиков
- необходимо иметь в виду требования заказчика и проект изменений API из предыдущего задания

C. Тесты Для поддержания тестового покрытия необходимо дополнить набор Unit и API тестов. Опишите в свободной форме для каких внедренных или измененных структурных единиц, кейсов и ответов API Вы бы подготовили тесты с кратким описанием их назначения.

Структуры корзины заказов

У нас есть набор действий, которые мы хотим осуществлять с заказами:

```
calculateTotalSum(){/*...*/}
getItems(){/*...*/}
getItemsCount(){/*...*/}
addItem($item){/*...*/}
deleteItem($item){/*...*/}
printOrder(){/*...*/}
showOrder(){/*...*/}
load(){/*...*/}
save(){/*...*/}
update(){/*...*/}
delete(){/*...*/}
```

Нужно создать структуру классов, чтобы можно было пользоваться этими методами.

Репозиторий билетов

Есть класс, который позволяет работать с билетами, находящимися в БД:

```
class TicketRepository
{
    public function load($ticketID)
    {
        return Ticket::find()->where(['id' => $ticketId])->one();
    }

    public function save($ticket){/*...*/}
    public function update($ticket){/*...*/}
    public function delete($ticket){/*...*/}

}
```

Стоит задача реализовать возможность работы с билетами которые хранятся на другом сервере (по API).

Требуется описать структуру методов и классов, с помощью которых можно будет загружать билеты как из БД, так и из другого сервера (по API).

Composer: Обновление зависимости

У вас есть проект, который использует библиотеку. Вам необходимо:

- внести изменения в библиотеку и протестировать ее работоспособность в проекте
- после успешного прохождения тестов вам необходимо выпустить релиз проекта с измененной библиотекой

Опишите Ваши действия в `git` и `composer` на всех этапах (разработка, тестирование, релиз и деплой).

SQL: Оценки студентов

A. Выборка данных Дана база данных с учениками и оценками.

Таблица `students` содержит очки студента (marks).

	id	name	marks
1	1	Liam	87
2	2	Olivia	65
3	3	Maria	95
4	4	James	64
5	5	Robert	83
6	6	John	95

Таблица `grade` - содержит соотношение очков к оценке студента.

	grade	min_mark	max_mark
1	1	0	9
2	2	10	19
3	3	20	29
4	4	30	39
5	5	40	49
6	6	50	59
7	7	60	69
8	8	70	79
9	9	80	89
10	10	90	100

Нужно написать запрос в базу данных, содержащий три столбца: `name`, `grade` и `mark`.

- данные должны быть в порядке убывания оценок, более высокие оценки выводятся первыми

- если есть более одного ученика с одинаковой оценкой (8-10), упорядочьте этих конкретных учеников по их именам в алфавитном порядке
- если оценка ниже 8, используйте “low” в качестве их имени и перечислите по оценкам в порядке убывания
- если есть более одного учащегося с одинаковой оценкой (1-7), упорядочите этих конкретных учащихся по их оценкам в порядке возрастания

B. Модификация DDL Предполагается, что таблица со студентами будет хранить данные свыше двух миллионов студентов и дальше будет только расти. Как бы вы модифицировали таблицы базы данных, при учете, что поле `grade` из таблицы `grade` всегда участвует в запросах к получению данных о студенте?

```
create table grade
(
    grade    int null,
    min_mark int null,
    max_mark int null
);

create table students
(
    id    int null,
    name  varchar(100) null,
    marks int null
);
```

Docker: Модификация конфигурации сервисов `docker-compose`

Есть конфигурация сервисов `docker-compose.yml`, требующая модификации. Необходимо:

- добавить в этот файл сервис где будет работать приложение `php`
- переопределить сервис базы данных на `mysql 8` не меняя текущий файл
- объединить все сервисы в одну сеть
- настройки портов и конфигурация сервиса `nginx` `default.conf` должны изменяться извне

```yaml
version: '3'
services:
    nginx:
        image: nginx:alpine
        container_name: app-nginx
        ports:
            - "8090:8090"
            - "443:443"
        volumes:
            - ./:/var/www
    db:
        platform: linux/x86_64
        image: mysql:5.6.47
        container_name: app-db
        ports:
            - "3306:3306"
        volumes:
            - ./etc/infrastructure/mysql/my.cnf:/etc/mysql/my.cnf
            - ./etc/database/base.sql:/docker-entrypoint-initdb.d/base.sql
```