Модуль 1 PHP и HTTP

Практическая №1.1: Проверка данных на стороне сервера

Скачайте, установите и настройте OpenServer. Скачайте раздаточный материал из личного кабинета и поместите его в новый хост second.site. Переименуйте файл с формой добавления книг formAddBook.html в formAddBook.php. Познакомьтесь с config.php и сделайте так, чтобы при отправке формы данные проверялись при помощи **filter_input()** добавлялись в таблицу и происходило перенаправление на сам же файл и при помощи сессий выводилось "flash" сообщение об удачном или неудачном добавлении. Реализуйте ссылку "Скачать", при нажатии на которую, происходило бы скачивание всех книг из базы в виде файла. Если будет время, реализуйте вывод всех книг под формой и встройте проверку при помощи регулярных выражений на наличие цифр в поле price

Модуль 2. Введение в ООП

Практическая №2.1: Создание классов

Создайте файл books.php и внутри классы **Goods**, **Book**, **Journal**, **IGoods**, **BookFabric, GoodsCollection.** В классе Book создайте свойства title, author, description, price и метод getHTML(), возвращающий информацию в виде HTML. Создайте два экземпляра класса Book с заполненными свойствами. Если чувствуете уверенность, можете поместить эти экземпляры в массив. Создайте файлы классов и перенесите код в соответствующие файлы. Реализуйте функцию для автозагрузки (коллбек для **spl_autoload_register()**). В классе Book создайте константы BOOK_HTML, BOOK_JSON, BOOK_CSV, BOOK_ARRAY со значениями HTML, JSON, CSV, ARRAY соответственно. В классе Book создайте конструктор, заполняющий свойства соответствующими аргументами title, author, description, price. Опишите в классе деструктор, он должен выводить "Книга [name] удалена
". Создайте 2 экземпляра класса Book и заполнить свойства. Добавьте метод get() в класс Book. Метод должен принимать аргумент $format с значением по умолчанию Book::BOOK_HTML. Выведите информацию по всем книгам

Практическая №2.2: Наследование

Добавьте классу Goods общедоступные свойства $title и $price, и конструктор, заполняющий свойства. Сделайте так, чтобы Book расширял класс Goods. Уберите из Book свойства, которые есть в Goods. Перепишите конструктор Book так, чтобы вызывался родительский конструктор parent::__constructor($title, $price). Перенесите константы из Book в Goods. Переименуйте их из BOOK_HTML (и др.) в GOODS_HTML. Обновите аргумент $format в методе get() класса Book. При необходимости, обработайте исключения.

Практическая №2.3: Абстрактные классы и интерфейсы

Создайте в классе Goods абстрактную функцию get(), а класс Goods отметьте абстрактным. Создайте интерфейс IGoods с общедоступными методами getHTML(), getCSV(), getJSON(), getArray(). Укажите выполнение классом Book интерфейса IGoods, Создайте в Book методы заглушки (без реализации) из IGoods. В классе Book сделайте финализированным метод getHTML(). Создайте тестовый класс расширяющий Book и перегружающий метод getHTML. В файл Journal.php скопируйте код класса Book и поменяйте обозначения на Journal (у нас появился новый класс, похожий на Book и расширяющий Goods - это может быть не совсем хорошая идея, но нам это нужно для последующей работы). Добавьте статические переменные $counter вашим классам Book и Journal. В файле books.php создайте экземпляр Journal. В конструкторах классов увеличивайте счетчики. Выведите количество книг и кол-во журналов. Откройте BookFabric.php и опишите одноимённый класс, в классе создайте статический метод get(), принимающий такие же аргументы как и конструктор класса Book. Создайте и верните экземпляр Book из метода get(). В файле books.php убедитесь, что получается создать экземляр Book через статический вызов BookFabric::get()

Практическая №2.4: Магические методы классов

Добавьте метод __clone() классу Book и вывести в нём фразу <hr>Клонирован экземпляр класса [CLASS]<hr>. В классе Book создайте "магический" метод __call() с двумя аргументами $name - принимает название вызываемого несуществующего метода и $arguments - массив аргументов, передаваемый вызываемому несуществующему методу. В теле __call() в произвольном виде выведите значения аргументов. Если будет время, сделайте так, чтобы __call() позволял делать вызовы типа $book->html() или $book->json(). Т.е. метод должен проверять существование соответствующего метода типа getHTML() и вызывать, если он существует. В файле books.php создайте экземпляр $gc класса GoodsCollection. Примечание: не забудьте передать массив книг и журналов в конструктор класса. Распечатайте в print_r() результат обращения к свойству Book объекта $gc. Откройте класс Book Опишите в классе метод __invoke(), опишите в классе метод __toString(). В файле books.php вызовите экземпляр Book как функцию. Распечатайте экземпляр класса Book через echo

Модуль 3. Работа с базами данных

Практическая №3.1: Работа с mysqli в объектно-ориентированном стиле.

Перепишите код formAddBook.php в объектно-ориентированном стиле. Обязательно используйте *подготовленные запросы*.

Модуль 4. ООП-реализация

Практическая №4.1: Знакомство с MVC

Познакомьтесь с вариантами реализации MVC в файлах раздачи. Создайте свою реализацию MVC.

Модуль 5. Composer

Практическая №5.1: Работа с composer

Убедитесь, что у вас есть composer или скачайте и установите его (https://www.youtube.com/watch?v=X-yrrl11qdE). Обновите composer self-update. Создайте файл composer.json при помощи команды composer init. Установите phpunit: composer require --dev phpunit/phpunit. Добавьте изменения в composer.json {"autoload": {"psr-0": {"Vendor\\Namespace": ["src/", "lib/"], "": "src/"} }, "config": { "optimize-autoloader": true } }. Выполните последовательно composer update и composer dump-autoload --optimize. В файле index.php укажите include "vendor/autoload.php". Создайте экземпляры произвольных классов из папки src (например, Car.php). Измените composer.json для psr: {"require-dev":{"phpunit/phpunit":"^9.3"},"autoload":{"psr-4":{"Application\\":"module/Application/src/","Vendor\\Namespace\\":""}},"config":{"optimize-autoloader":true}} и перенесите классы в папку module/Application/src/. В файле самих классов укажите пространство имён типа namespace Application\Car, а в index.php создайте экземпляр new \Application\Book\Book(). После composer update, проверьте работу приложения в браузере. Познакомьтесь с https://packagist.org/ и установите произвольные пакеты (например Monolog или Faker). Если будет время, установите Laravel или Symfony (https://www.youtube.com/watch?v=ABdeoIm6e74)

Модуль 6. Вспомогательные инструменты

Практическая №6.1: Тестирование, анализ и документирование кода

Напишите тестовый класс BookTest для класса Book и расположите его в папке module/Application/tests. Перейдите в папку с PHPUnit: cd vendor/phpunit/phpunit и проверьте версию фреймворка для тестирования php phpunit --version. Вернитесь в папку проекта и выполните тесты php vendor/phpunit/phpunit/phpunit --bootstrap vendor/autoload.php module/Application/tests. Тестовый файл будет примерно таким:

```
<?php
use PHPUnit\Framework\TestCase;
use Application\Book\Book;

class BookTest extends TestCase
{
   public function testNewBook()
    {
       $newBook = new Book();
       $this->assertSame( get_class($newBook) , 'Application\Book\Book');
    }
}
```

Установите PHPStan: composer require --dev phpstan/phpstan (или Psalm, или Phan). Выполните статические анализ кода при помощи PHPStan: php vendor/phpstan/phpstan/phpstan analyze module/Application/src module/Application/tests. Выберите уровень анализа https://phpstan.org/user-guide/rule-levels и проанализируйте свои классы php vendor/phpstan/phpstan/phpstan analyze -l 4 module/Application. Попробуйте повысить уровень и вывести результаты в файл php vendor/phpstan/phpstan/phpstan analyze -l 8 --error-format=prettyJson module/Application > errors.json. Скачайте одну из последних версий PHPDoc (в формате phar): https://github.com/phpDocumentor/phpDocumentor/releases/. Используя PHPDoc синтаксис, напишите комментарии к классу Book и сгенерируйте документацию при помощи PHPDoc: php phpDocumentor.phar -d ./module/Application/src/. Подсказка: посмотрите теги https://docs.phpdoc.org/latest/references/phpdoc/index.html

```
<?php
 /**
  * @author My Name
  * @author My Name <my.name@example.com>
  */
```

```
namespace Application\Book;

/**
 * I belong to a class
 */
class Book {
/**
 * @param string $argument1 This is the description.
 * @return string  This is the description.
 */
public function foo($str){

 }
}
```

Модуль 7. Создание интернет магазина

Практическая №7.1: Итоговая работа