

# Лабораторные работы по курсу: PHP8 часть 4.



## Профессиональная работа

### Модуль 1. Продвинутое возможности (4 ак. ч.)

#### Лабораторная 1.1 Работа с объектно-ориентированными возможностями по теме модуля

1. Создайте переменную на основе `new \DateTime()`
2. С помощью метода `setTimestamp()` установите количество секунд, прошедших от 1970 до 12 апреля 1961
3. С помощью метода `setTimezone` установите часовой пояс `DateTimeZone::EUROPE`
4. Отформатируйте дату через метод `format('Y/m/d H:i:s')` и выведите данные

#### Лабораторная 1.2 Реализация автозагрузки

1. Познакомьтесь с примером  
  Composer автозагрузка своих php-классов psr-4
2. Реализуйте PSR-4 автозагрузку своих файлов через composer

## Модуль 2. Расширенные методы (4 ак. ч.)

### Лабораторная 2.1 Реализация интерфейса кэширования

1. Познакомьтесь с <https://symfony.com/doc/current/components/cache.html>
2. Установите модуль `composer require symfony/cache` и реализуйте произвольную логику по кэшированию данных
3. \* В качестве задания для самостоятельной работы: реализуйте классы, выполняющие интерфейсы PSR-6 <https://www.php-fig.org/psr/psr-6/>

### Лабораторная 2.2 Реализация другого интерфейса

1. Познакомьтесь с <https://packagist.org/packages/guzzlehttp/guzzle>
2. Установите модуль `composer require guzzlehttp/guzzle`
3. Реализуйте фрагмент кода, выполняющего GET или POST запрос к <https://jsonplaceholder.typicode.com/>

# Модуль 3. PSR-7 Обмен сообщений и Middleware (4 ак. ч.)

## Лабораторная Создание Middleware

1. Установите фреймворк Slim и его компонент PSR-7  
(<http://www.slimframework.com/docs/v4/start/installation.html>)

```
composer require slim/slim:"4.*"  
composer require slim/psr7
```

2. Создайте index.php

```
use Psr\Http\Message\ResponseInterface as Response;  
use Psr\Http\Message\ServerRequestInterface as Request;  
use Slim\Factory\AppFactory;  
require 'vendor/autoload.php';  
$app = AppFactory::create();  
$app->get('/', function (Request $request, Response $response, $args) {  
    $response->getBody()->write("Hello world!");  
    return $response;  
});  
$app->run();
```

3. Создайте Middleware и посмотрите результат в работе

```
use Psr\Http\Message\ServerRequestInterface as Request;  
use Psr\Http\Server\RequestHandlerInterface as RequestHandler;  
use Slim\Factory\AppFactory;  
use Slim\Psr7\Response;  
require 'vendor/autoload.php';  
$app = AppFactory::create();  
$app->add(function (Request $request, RequestHandler $handler) {  
    $response = $handler->handle($request);  
    $existingContent = (string) $response->getBody();  
    $response = new Response();  
    $response->getBody()->write('BEFORE ' . $existingContent);  
    return $response;  
});  
$app->add(function (Request $request, RequestHandler $handler) {  
    $response = $handler->handle($request);
```

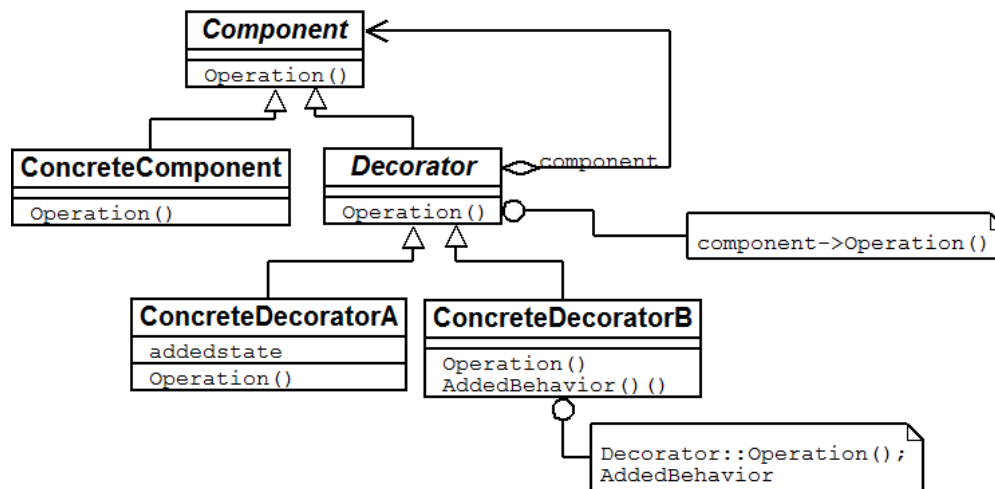
```
        $response->getBody()->write(' AFTER');  
        return $response;  
    });  
    $app->get('/', function (Request $request, Response $response, $args)  
    {  
        $response->getBody()->write("Hello world!");  
        return $response;  
    });  
    $app->run();
```

4. (Оptionальная часть для самостоятельной работы) Изучите и используйте другие middleware

```
composer require slim/twig-view  
composer require bryanjhv/slim-session:~4.0
```

## Модуль 4. Архитектура приложения (4 ак. ч.)

### Лабораторная: Декоратор



1. Изучите реализацию шаблона Декоратор

<?php

```
class RequestHelper {}
```

```
abstract class ProcessRequest {
    abstract function process ( RequestHelper $req );
}
```

```
class MainProcess extends ProcessRequest {
    function process ( RequestHelper $req ) {
        print __CLASS__ . ": выполнение запроса\n<br>";
    }
}
```

```
abstract class DecorateProcess extends ProcessRequest {
    protected $processrequest;

    function __construct ( ProcessRequest $pr ) {
        $this->processrequest = $pr;
    }
}
```

```
class LogRequest extends DecorateProcess {
    function process ( RequestHelper $req ) {
```

```

        print __CLASS__ . ": регистрация запроса \n<br>";
        $this->processrequest->process( $req );
    }
}
class AuthenticateRequest extends DecorateProcess {
    function process ( RequestHelper $req ) {
        print __CLASS__ . ": аутентификация запроса \n<br>";
        $this->processrequest->process( $req );
    }
}
class StructureRequest extends DecorateProcess {
    function process ( RequestHelper $req ) {
        print __CLASS__ . ": упорядочение данных запроса \n<br>";
        $this->processrequest->process( $req );
    }
}

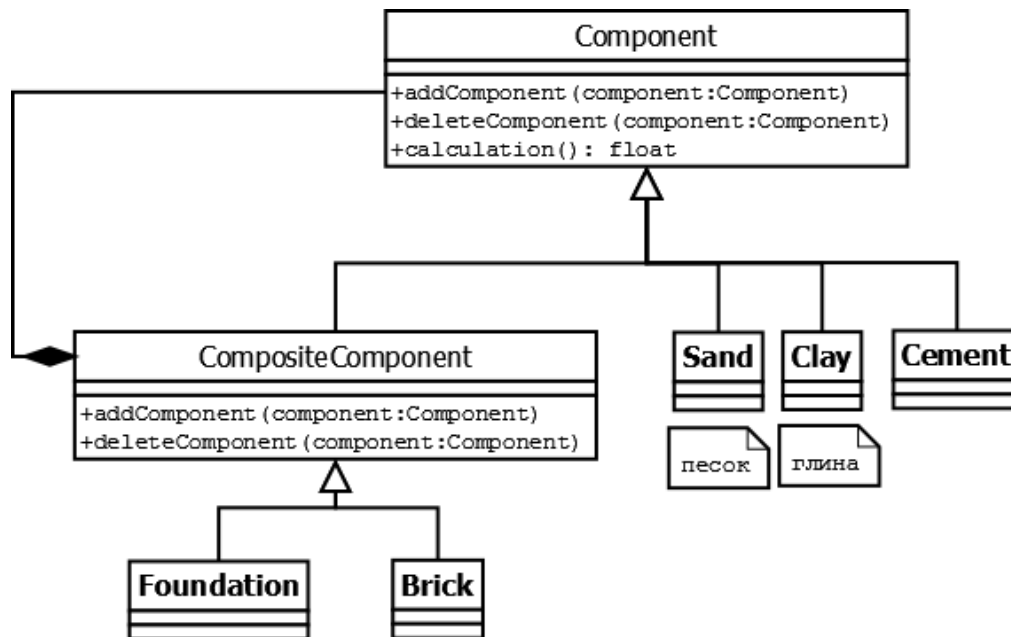
$process= new AuthenticateRequest (
            new StructureRequest (
                new MainProcess ()
            )
        );

$process->process( new RequestHelper );

```

## 2. Создайте пример шаблона Декоратор

## Лабораторная: Composite



1. Познакомьтесь с реализацией шаблона Composite

<?php

```
abstract class Component{
    abstract function addComponent(Component $component);
    abstract function deleteComponent(Component $component);
    abstract function calculation();
}

class Sand extends Component{
    function calculation()
    {
        return 3;
    }
    function addComponent(Component $component){ }
    function deleteComponent(Component $component){ }
}

class Clay extends Component{
    function calculation()
    {
        return 5;
    }
    function addComponent(Component $component){ }
    function deleteComponent(Component $component){ }
}
```

```

class Cement extends Component{
    function calculation()
    {
        return 15;
    }
    function addComponent(Component $component){ }
    function deleteComponent(Component $component){ }
}
class Foundation extends Component{
    private $components = [];
    function addComponent(Component $component){
        if( in_array( $component, $this->components, true)){
            return;
        }
        $this->components[] = $component;
    }
    function deleteComponent(Component $component){
        $this->components = array_udiff($this->components,[$component],
            function($a, $b){return ($a === $b)? 0 : 1;});
    }
    function calculation()
    {
        $tmp = 0;
        foreach($this->components as $component){
            $tmp += $component->calculation();
        }
        return $tmp * 2 + 1000;
    }
    function pr(){
        echo "<pre>";
        print_r($this->components[0]);
        echo "</pre>";
    }
}
class Brick extends Component{
    private $components = [];
    function addComponent(Component $component){
        if( in_array( $component, $this->components, true)){
            return;
        }
        $this->components[] = $component;
    }
    function deleteComponent(Component $component){
        $this->components = array_udiff($this->components,[$component],
            function($a, $b){return ($a === $b)? 0 : 1;});
    }
}

```



```

    }
    function calculation()
    {
        $tmp = 0;
        foreach($this->components as $component){
            $tmp += $component->calculation();
        }
        return $tmp * 1.5;
    }
    function pr(){
        echo "<pre>";
        print_r($this->calculation());
        echo "</pre>";
    }
}

$foundation = new Foundation();
$foundation->addComponent(new Sand());
$foundation->addComponent(new Cement());
echo "Стоимость фундамента: ", $foundation->calculation(), "<br>";

$brick = new Brick();
$brick->addComponent(new Clay());
echo "Стоимость кирпича: ", $brick->calculation(), "<br>";

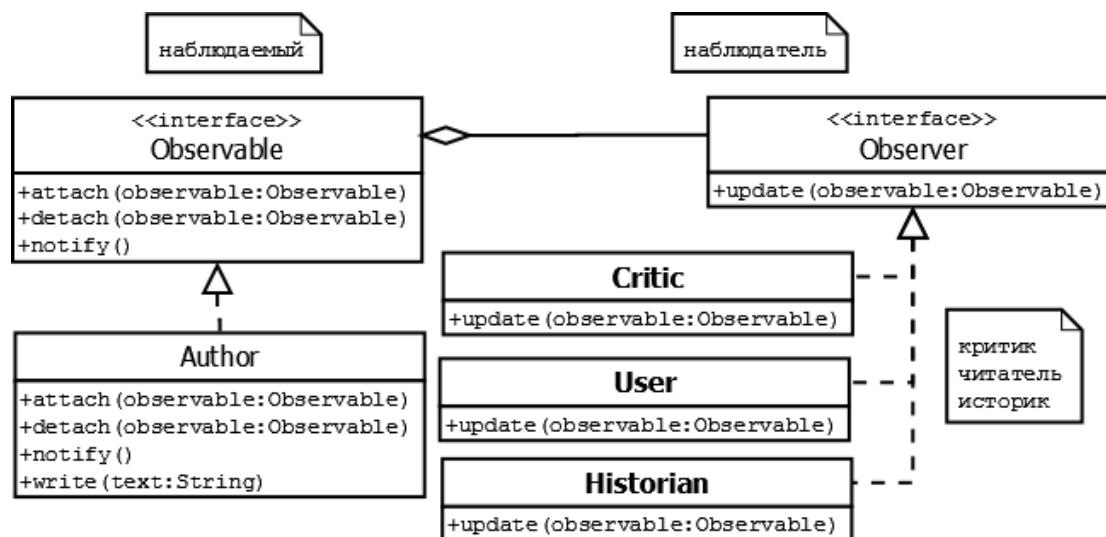
$foundation->addComponent($brick);

echo "Стоимость фундамента с кирпичом: ",
$foundation->calculation(), "<br>";

```

## 2. Реализуйте шаблон Composite

## Лабораторная: Observer/Наблюдатель



1. Познакомьтесь с реализацией шаблона Observer/Наблюдатель

<?php

```

interface Observable {
    function attach(Observer $observer ) ;
    function detach(Observer $observer );
    function notify();
}

class Author implements Observable{
    private $observers=[];
    public $name = "";
    function __construct($name){
        $this->name = $name;
        $this->observers = [];
    }
    function attach (Observer $observer ) {
        $this->observers[] = $observer;
    }
    function detach (Observer $observer ) {
        $this->observers = array_filter($this->observers,function ( $a )
use ( $observer ) {
        return ( ! ( $a === $observer ) ) ;
    });
    }
    function write($text){
        echo $this->name," написал: $text<br>";
        $this->notify();
    }
}
    
```

```

        function notify(){
            foreach($this->observers as $obs)
                $obs->update($this);
        }
    }

    interface Observer{
        function update( Observable $observable );
    }

    class Critic implements Observer{
        public $name = "";
        function __construct($name){
            $this->name = $name;
        }
        function update( Observable $observable ){
            echo "Критик ", $this->name, " написал: наконец-то
<b>{$observable->name}</b> что-то написал..

```

```
$user = new User("Г. Сумкин");  
$user2 = new User("А. Рогова");  
$historian = new Historian("Д.И Иванов");  
  
$author->attach($critic);  
$author->attach($user);  
$author->attach($user2);  
$author->attach($historian);  
  
$author->write("Когда для смертного умолкнет шумный день..");  
  
$author->detach($critic);  
$author->write("Воспоминание");
```

## 2. Реализуйте шаблон Observer/Наблюдатель

## Лабораторная Singleton/Одиночка

1. Познакомьтесь с реализацией шаблона Singleton/Одиночка и реализуйте свой вариант

```
<?php
```

```
class Preferences {
    private $props = array();
    private static $instance;

    private function __construct() { }

    public static function getInstance() {
        if( empty( self::$instance ) ) {
            self::$instance = new Preferences;
        }
        return self::$instance;
    }

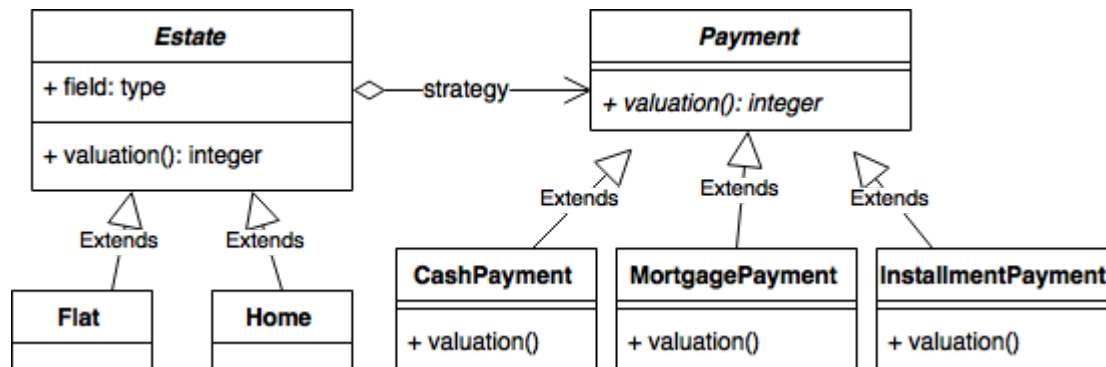
    public function setProperty( $key, $val ) {
        $this->props[$key] = $val;
    }

    public function getProperty( $key ) {
        return $this->props[$key];
    }
}

$pref = Preferences::getInstance();
$pref->setProperty( "name", "Иван" );
unset( $pref );
$pref2 = Preferences::getInstance();
echo $pref2->getProperty( "name" );

?>
```

## Лабораторная Strategy/Стратегия



1. Познакомьтесь с шаблоном Стратегия и его реализацией

<?php

```
abstract class Estate {
    protected $address = "";
    protected $price = "";
    protected $strategy;

    function __construct ( $address, $price, Payment $strategy){
        $this->address = $address;
        $this->price = $price;
        $this->strategy = $strategy;
    }

    function valuation(){
        return $this->strategy->valuation($this->price);
    }
}

class Flat extends Estate{

}

class Home extends Estate{

}

abstract class Payment {
    abstract function valuation($price);
}

class CashPayment extends Payment{
```

```

    function valuation($price){
        return $price;
    }
}

class MortgagePayment extends Payment {
    private $firstPayment;
    private $p;
    private $n;
    function __construct($firstPayment, $p, $n){
        $this->firstPayment = $firstPayment;
        $this->p = $p / 1200;
        $this->n = $n * 12;
    }
    function valuation($price){
        return ceil($this->n * ($price - $this->firstPayment)* $this->p/ (1
- pow(1 + $this->p, -$this->n)));
    }
}

class InstallmentPayment extends Payment {
    function __construct($p){
        $this->p = $p;
    }

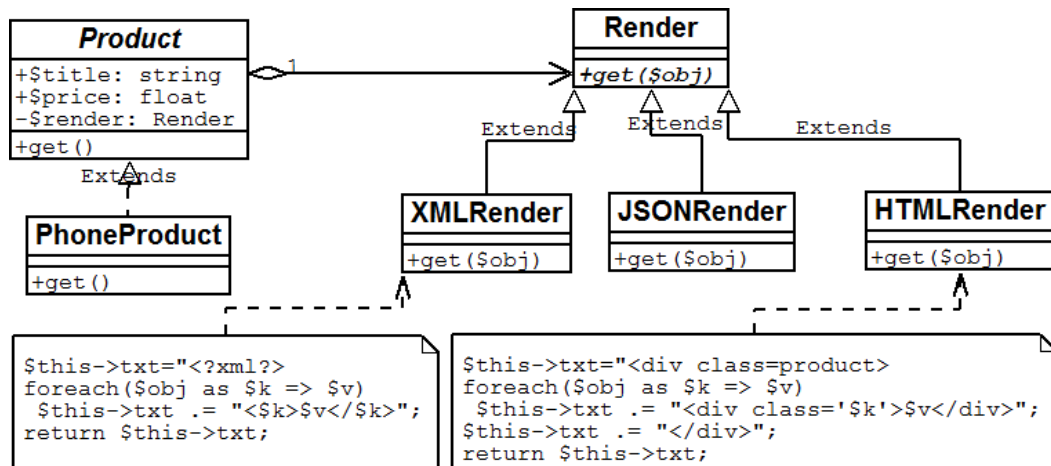
    function valuation($price){
        return ceil($price * (1 + $this->p/100));
    }
}

$strategies = array(
    new CashPayment(),
    new MortgagePayment(100000, 12, 10),
    new InstallmentPayment(1)
);

foreach($strategies as $strategy){
    $flat2room = new Flat("Люберцы", 5e6, $strategy);
    echo $flat2room->valuation(), "<br>";
}

```

2. Сделайте реализацию шаблона Стратегия на примере следующей UML-диаграммы





## Лабораторная работа 4.2 Реализация произвольного шаблона

1. Познакомьтесь с примерами других шаблонов <https://refactoring.guru/>
2. Реализуйте свой вариант одного из шаблонов

## Модуль 5. Reflection (2 ак. ч.)

### Лабораторная работа 5.1 Работа с Reflection API

1. Создайте класс Some.php

```
<?php

/**
 * Просто класс
 */
class Some {
    public static $counter = 1;

    public $test = "test";

    public function __construct(){
        echo "я конструктор<br>";
    }
    public function someFunc(Array $arr = [], $d = 45){
        return "test$d";
    }
}
```

2. При помощи инструментов Reflection получите информацию о классе Some

```
<?php
include "Some.php";
$refclass = new ReflectionClass("Some");
echo "<pre>";
Reflection::export($refclass);
```

3. Получите информацию о методах класс Some

```
<?php
include "Some.php";
$refclass = new ReflectionClass("Some");
```

```

$methods = get_class_methods("ReflectionClass");
echo "<pre>";
foreach($methods as $method){
    echo "$method: ",$refclass->$method(),"<br>";
}
echo "</pre>";

```

4. Получите сведения о модификаторах доступа и другой информации

```

<?php
include "Some.php";
$refclass = new ReflectionClass("Some");

$methods = $refclass->getMethods();

$props = [
    "isUserDefined",
    "isInternal",
    "isPublic",
    "isAbstract",
    "isProtected",
    "isPublic",
    "isPrivate",
    "isStatic",
    "isFinal",
    "isConstructor",
];

echo "<pre>";
//print_r($methods);
foreach($methods as $method){
    echo "<hr>Метод: ",$method->getName(),"<br>";
    foreach($props as $prop){
        echo "$prop: ",$method->$prop(),"<br>";
    }
}
echo "</pre>";

```

5. Получите информацию о параметрах метода someFunc

```

<?php

```

```
include "Some.php";
$refclass = new ReflectionClass("Some");

$method = $refclass->getMethod("someFunc");
$parameters = $method->getParameters();

$props = [
    "allowsNull",
    "getDefaultValue",
    "getName",
    "getPosition",
    "isArray",
    "isCallable",
];

echo "<pre>";
//print_r($methods);
foreach($parameters as $parameter){
    echo "<hr>Аргумент: ", $parameter->getName(), "<br>";
    foreach($props as $prop){
        echo "$prop: ", $parameter->$prop(), "<br>";
    }
}
echo "</pre>";
```

# Модуль 6. PDO (PHP Data Objects) (2 ак. ч.)

## Лабораторная работа 6.1 Работа с PDO

1. Создайте базу данных SQLite3 при помощи конструктора PDO, назовите её

```
sqlite:mydb.sqlite3
```

```
$user = "root";  
$pass = "";  
$dbh = new PDO('sqlite:mydb.sqlite3', $user, $pass);
```

2. Создайте в базе данных таблицу user с полями текстового типа host и user

```
$dbh->exec("CREATE TABLE user(host TEXT,user TEXT);");
```

3. Выполните вставку данных в базу двух записей: ('localhost', 'Ivan') и ('localhost', 'Vasiliy')

```
try{  
    $sql = "INSERT INTO user VALUES('localhost','Ivan')";  
    $dbh->exec($sql)or die(print_r($dbh->errorInfo(), true));  
    $sql = "INSERT INTO user VALUES('localhost','Vasiliy')";  
    $dbh->exec($sql);  
}catch (PDOException $e) {  
    print "Error!: " . $e->getMessage() . "<br/>";  
    die();  
}
```

4. Выполните запрос на выборку данных из базы

```
foreach($dbh->query('SELECT user,host from user') as $row) {  
    echo "<pre>";  
    print_r($row);  
    echo "</pre><hr>";  
}
```

5. Убедитесь, что есть доступ к базе данных MySQL и выполните запрос на получение данных о хостах и пользователях встроенной базы user. Примечание: учётные данные пользователя - логин и пароль - поместите в переменные \$user и \$pass

```

$dbh = new PDO('mysql:host=localhost;dbname=mysql', $user, $pass);

foreach($dbh->query('SELECT user,host from user') as $row) {
    echo "<pre>";
    print_r($row);
    echo "</pre><hr>";
}

```

6. Получите данные из базы MySQL в виде ассоциативного массива

```

$stmt = $dbh->query('SELECT user,host from user');
while($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
    echo "<pre>";
    print_r($row);
    echo "</pre><hr>";
}

```

7. Получите данные из базы MySQL в виде объектов класса User

```

$dbh = new PDO('mysql:host=localhost;dbname=mysql', $user, $pass);

$stmt = $dbh->query('SELECT user,host from user');
$arrObj= $stmt->fetchAll(PDO::FETCH_CLASS, 'User');
foreach ($arrObj as $obj){
    echo $obj-> getParams(),"<hr>";
}

class User {
public $user, $host;
function getParams(){
return ($this->user."|".$this->host);
}
}

```

8. Выполните подготовленный запрос, который найдёт в базе MySQL всех пользователей с названием хоста '%'

```

$host = '%';
$stmt = $dbh->prepare('SELECT user,host
    FROM user
    WHERE host = :host');
$stmt->bindParam(':host', $host,PDO::PARAM_STR,1);

```

```
$sth->execute();
```

```
$result = $sth->fetchAll();  
echo "<pre>", print_r($result), "</pre>";
```

9. \* (Это задание рекомендуется к выполнению тем, кто проходил курс по MySQL  
8) Создайте и вызовите хранимую процедуру

```
/*  
delimiter //  
DROP PROCEDURE IF EXISTS sp_return//  
CREATE PROCEDURE sp_return (OUT n INT(5))  
BEGIN  
    SELECT 123 INTO n;  
END//  
delimiter ;  
*/  
  
$stmt = $dbh->prepare("CALL sp_return(@num)");  
$stmt->bindParam("@num", $return_value) ;  
  
// вызов хранимой процедуры  
$stmt->execute() ;  
  
print "процедура вернула $return_value\n";
```

10. \* (Это задание рекомендуется к выполнению тем, кто проходил курс по MySQL  
8) Создайте транзакцию на одновременное добавление двух записей в таблицу

```
try {  
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);  
  
    $dbh->beginTransaction();  
    $dbh->exec("insert into user (user,host) values ('Joe', 'somehost')");  
    $dbh->exec("insert into user (user,host) values ('John', 'foohost')");  
  
    $dbh->commit();  
}catch (Exception $e) {  
    $dbh->rollBack();  
    echo "Ошибка: " . $e->getMessage();  
}
```

# Модуль 7. Окружение сервера (2 ак. ч.)

## Лабораторная работа 7.1 Использование инструментов

1. Установите Phing через composer.json

```
{  
    "require-dev": {  
        "phing/phing": "3.*"  
    },  
    "minimum-stability": "alpha"  
}
```

или (предпочтительней) скачайте phar-файл с <https://www.phing.info/> и переименуйте его в phing.phar

2. Убедитесь, что доступен вызов `php phing.phar`, в ответ будет выдано сообщение "Buildfile: build.xml does not exist!"
3. Создайте файл build.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<project name="FooBar" default="dist">
```

```
</project>
```

4. Внутри тегов <project> укажите создание директории для сборки build

```
<target name="prepare">  
    <echo msg="Making directory ./build" />  
    <mkdir dir="./build" />  
</target>
```

5. Следом за созданием директории сборки, укажите какие именно файлы попадут в сборку (в примере в сборку добавляется файл some.php - обязательно укажите ещё несколько других файлов отдельным элементом <copy>)

```
<target name="build" depends="prepare">  
    <echo msg="Copying files to build directory..." />  
  
    <echo msg="Copying ./some.php to ./build directory..." />  
    <copy file="./some.php" tofile="./build/some.php" />
```

```
</target>
```

6. Создайте отдельную задачу по архивированию скопированных файлов в архив build.tar.gz

```
<target name="dist" depends="build">
    <echo msg="Creating archive..." />

    <tar destfile="./build/build.tar.gz" compression="gzip">
        <fileset dir="./build">
            <include name="*" />
        </fileset>
    </tar>

    <echo msg="Files copied and compressed in build directory OK!"
/>
</target>
```

7. Выполните сборку командой php phing.phar
8. Убедитесь, что архив build.tar.gz содержит добавленные файлы
9. Добавьте содержимое произвольной папки в zip-архив

```
<zip destfile="phing.zip">
    <fileset dir=".">
        <include name="**/**" />
    </fileset>
</zip>
```

10. Укажите уведомление по емейл о сборке (тестовый пример, без настроенного SMTP работать не будет)

```
<mail tolist="user@mysite.local" subject="build complete"
from="test@mysite.local">
    The build process is a success...
</mail>
```

11. Используйте элемент <input> для получения данных от пользователя в момент сборки

```
<echo msg="questions..." />
```



```
<echo>My favorite format: ${format}</echo>
```

```
<input propertyname="format"  
defaultValue="json" promptChar="?">Please, select favorite  
format</input>
```

```
<echo>My favorite format: ${format}</echo>
```

```
<echo>Choose a valid option:</echo>
```

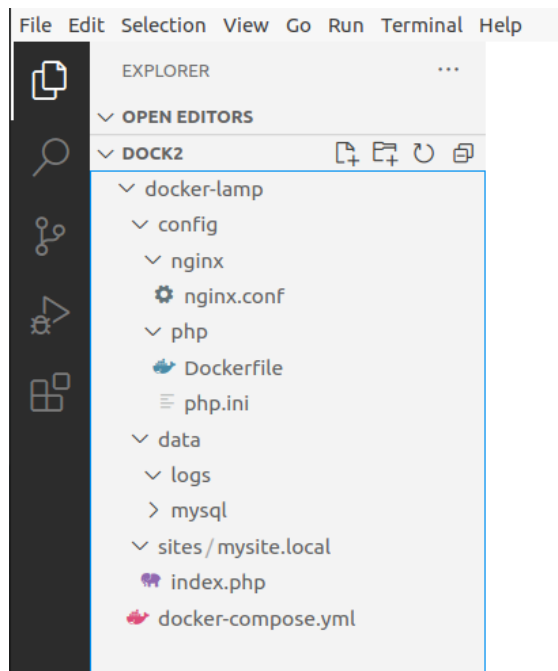
```
<input propertyname="optionsChoice" validargs="json,xml,yaml">  
Which item would you like to use  
</input>
```

12. Выполните сборку командой `php phing.phar`

# Модуль 8. Docker (2 ак. ч.)

## Лабораторная работа 8

1. Скачайте и установите Docker (работа показана под Ubuntu)
2. Создайте файловую структуру



3. В index.php укажите

```
<?php  
phpinfo();
```

4. В DockerFile укажите

```
FROM php:7.3-fpm  
  
RUN apt-get update && apt-get install -y \  
    curl \  
    wget \  
    git \  
    libfreetype6-dev \  
    libjpeg62-turbo-dev \  
    libmcrypt-dev \
```

```

        libpng-dev \
        libzip-dev \
        && docker-php-ext-install -j$(nproc) iconv mbstring mysqli
pdo_mysql zip \
        && docker-php-ext-configure gd
--with-freetype-dir=/usr/include/ --with-jpeg-dir=/usr/include/ \
        && docker-php-ext-install -j$(nproc) gd

RUN curl -sS https://getcomposer.org/installer | php --
--install-dir=/usr/local/bin --filename=composer

USER www-data:www-data

WORKDIR /var/www
CMD ["php-fpm"]

```

## 5. В docker-compose.yml укажите

```

version: '3'

networks:
  internal:

services:
  nginx:
    image: nginx:stable-alpine
    container_name: nginx
    ports:
      - "80:80"
    volumes:
      - ./sites:/var/www
      - ./config/nginx:/etc/nginx/conf.d
      - ./data/logs:/var/log/nginx/
    depends_on:
      - php
      - mysql
    networks:
      - internal
  php:

```

```

build:
  context: ./config/php
  dockerfile: Dockerfile
container_name: php
volumes:
  - ./sites:/var/www
  - ./config/php/php.ini:/usr/local/etc/php/php.ini
ports:
  - "9000:9000"
networks:
  - internal

mysql:
  image: mysql:5.7
  container_name: mysql
  restart: unless-stopped
  command:
--default-authentication-plugin=mysql_native_password
  command: --innodb_use_native_aio=0
  ports:
    - "3306:3306"
  volumes:
    - ./data/mysql:/var/lib/mysql
  environment:
    MYSQL_ROOT_PASSWORD: secret
  networks:
    - internal

```

## 6. В nginx.conf укажите

```

server {
    index index.php;
    server_name mysite.local;
    error_log /var/log/nginx/error.log;
    access_log /var/log/nginx/access.log;
    root /var/www/mysite.local;

    location ~ /\.php$ {
        try_files $uri =404;
        fastcgi_split_path_info ^(.+\.(php))(/.+)$;
    }
}


```

```

        fastcgi_pass php:9000;
        fastcgi_index index.php;
        include fastcgi_params;
        fastcgi_param SCRIPT_FILENAME
    $document_root$fastcgi_script_name;
        fastcgi_param PATH_INFO $fastcgi_path_info;
    }
}

```

7. В файле hosts добавьте описание сайта: 127.0.0.1 mysite.local (/etc/hosts)
8. В консоли, в папке docker-lamp, создайте контейнеры: `docker-compose up -d`
9. Перейдите в браузер и посмотрите на заставку с данными о php!



<b>System</b>	Linux 9ee3e56802b5 5.4.0-86-generic #97-Ubuntu
<b>Build Date</b>	Sep 28 2021 16:43:33
<b>Configure Command</b>	'./configure' '--build=x86_64-linux-gnu' '--with-config-scan-dir=/usr/local/etc/php/conf.d' '--enable-option-enable-ftp' '--enable-mbstring' '--enable-mysqlnd' '--with-pdo-sqlite=/usr' '--with-sqlite3=/usr' '--with-curl' '--with-libdir=lib/x86_64-linux-gnu' '--enable-fpm' '--with-data' '--disable-cgi' 'build_alias=x86_64-linux-gnu'
<b>Server API</b>	FPM/FastCGI
<b>Virtual Directory Support</b>	disabled
<b>Configuration File (php.ini) Path</b>	/usr/local/etc/php
<b>Loaded Configuration File</b>	/usr/local/etc/php/php.ini
<b>Scan this dir for additional .ini files</b>	/usr/local/etc/php/conf.d
<b>Additional .ini files parsed</b>	/usr/local/etc/php/conf.d/docker-php-ext-gd.ini, /usr/local/etc/php/conf.d/docker-php-ext-pdo_mysql.sodium.ini, /usr/local/etc/php/conf.d/docker-php-ext-
<b>PHP API</b>	20180731