

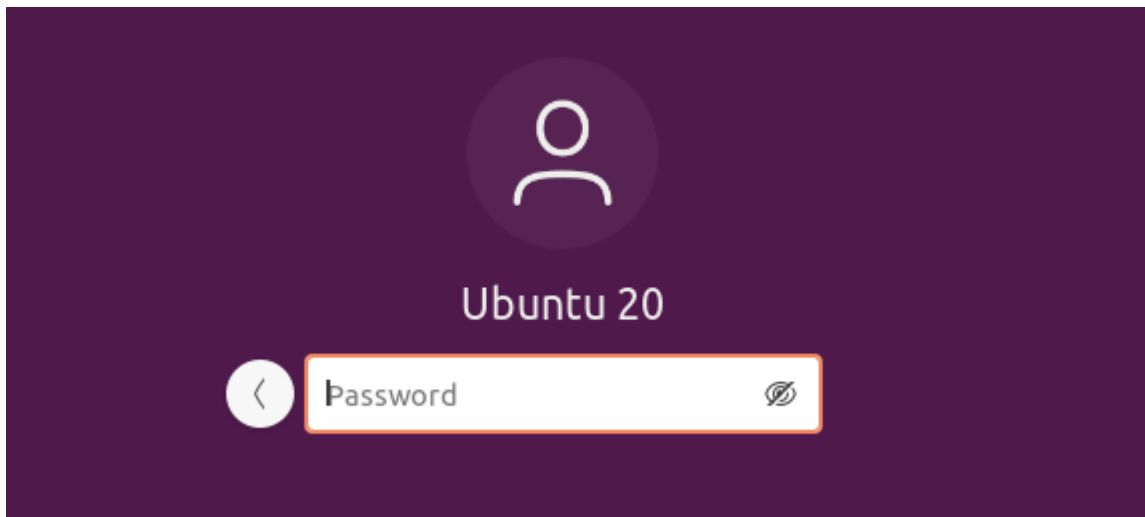
# Лабораторная работа №1

## PHP и его экосистема

### Теоретическая часть

#### Настройка окружения

#### Запуск VM Ubuntu 20



user: miet

password: miet

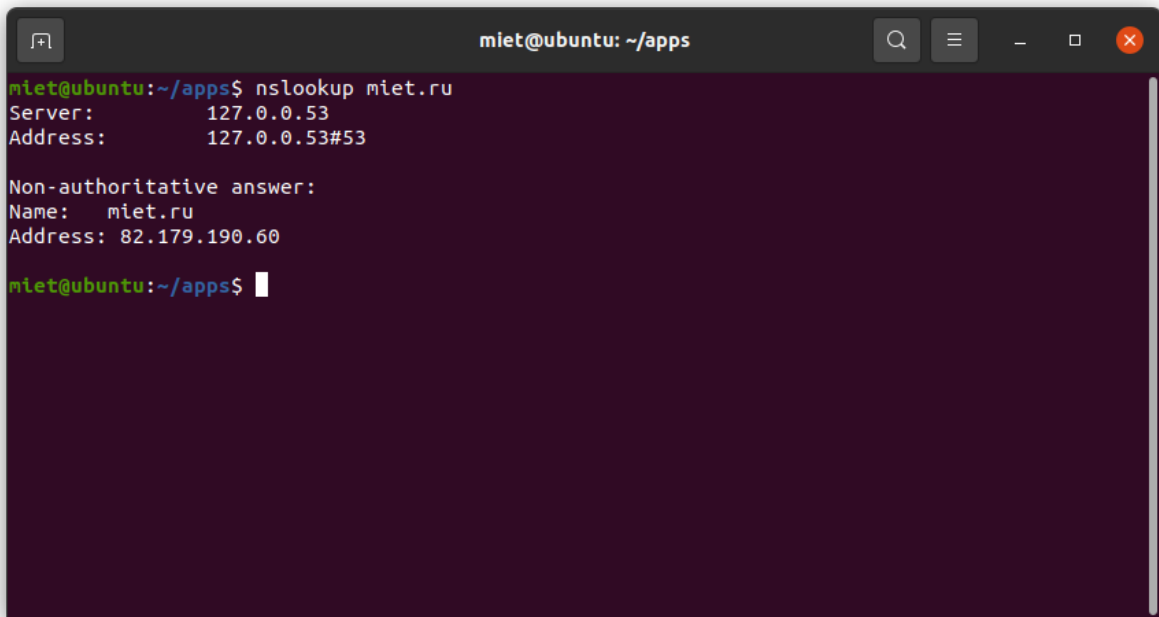
#### Список ПО на VM

Firefox  
Postman  
VS Code  
pgAdmin  
git  
Yarn  
NodeJS  
PHP 8.0

composer  
laravel installer  
PostgreSQL  
Elasticsearch  
Java  
Docker  
nginx

## Системные утилиты сетевой диагностики

**nslookup** – отображает сведения, которые можно использовать для диагностики инфраструктуры системы доменных имен (DNS). Перед использованием этого средства необходимо ознакомиться с принципами работы DNS.



```
miet@ubuntu: ~/apps
miet@ubuntu:~/apps$ nslookup miet.ru
Server:      127.0.0.53
Address:     127.0.0.53#53

Non-authoritative answer:
Name:   miet.ru
Address: 82.179.190.60

miet@ubuntu:~/apps$
```

**whois** – это сервис, который позволяет узнать всю информацию о владельце домена.

```
miet@ubuntu: ~/apps
miet@ubuntu:~/apps$ whois miet.ru
% By submitting a query to RIPN's Whois Service
% you agree to abide by the following terms of use:
% http://www.ripn.net/about/servpol.html#3.2 (in Russian)
% http://www.ripn.net/about/en/servpol.html#3.2 (in English).

domain:            MIET.RU
nserver:           ns2.miet.ru. 82.179.190.64
nserver:           ns3.miet.ru. 82.179.181.5
nserver:           ns.miet.ru. 82.179.191.64
state:             REGISTERED, DELEGATED, VERIFIED
org:               "National research university "Moscow Institute of Electronic Equipment"
registrar:         RU-CENTER-RU
admin-contact:     https://www.nic.ru/whois
created:           2000-09-18T20:00:00Z
paid-till:         2022-09-20T21:00:00Z
free-date:         2022-10-22
source:            TCI

Last updated on 2021-12-10T09:51:30Z

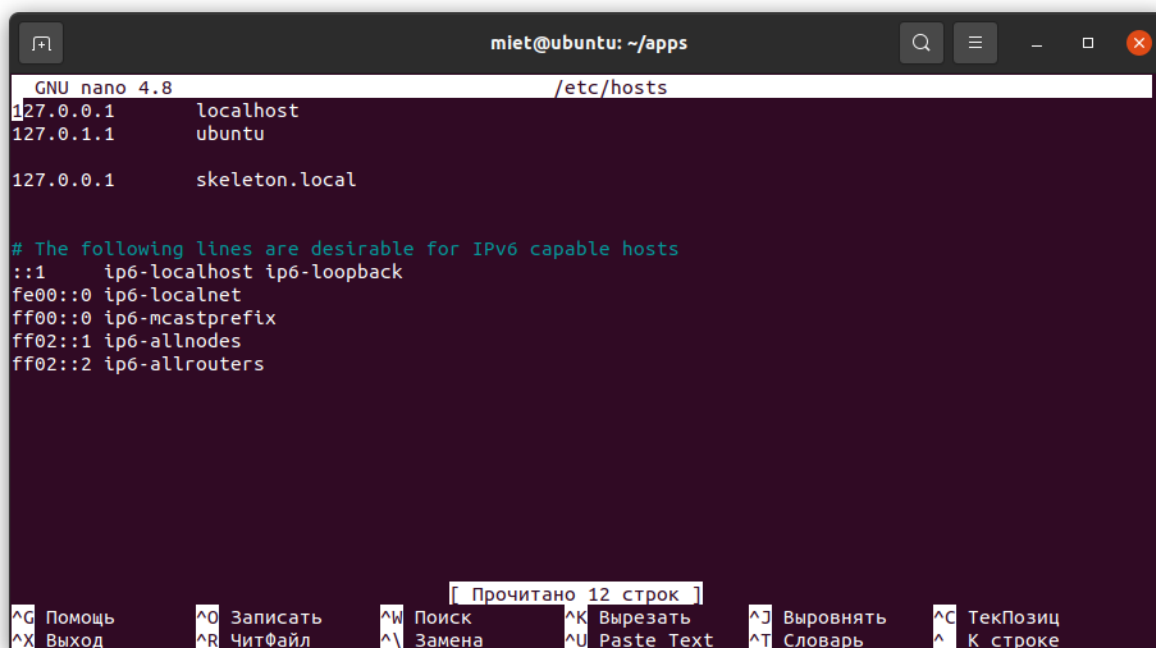
miet@ubuntu:~/apps$
```

## Настройка системы именования DNS

С помощью файла `/etc/hosts` можно настроить соответствие между доменом и конкретным IP-адресом.

Редактировать файл `/etc/hosts`:

```
sudo nano /etc/hosts
```

A terminal window titled 'miet@ubuntu: ~/apps' shows the nano 4.8 editor editing the /etc/hosts file. The file content is as follows:

```
127.0.0.1    localhost
127.0.1.1    ubuntu
127.0.0.1    skeleton.local

# The following lines are desirable for IPv6 capable hosts
::1         ip6-localhost ip6-loopback
fe00::0     ip6-localnet
ff00::0     ip6-mcastprefix
ff02::1     ip6-allnodes
ff02::2     ip6-allrouters
```

The status bar at the bottom indicates 'Прочитано 12 строк' (Read 12 lines) and lists various keyboard shortcuts like '^G Помощь', '^O Записать', etc.

Добавьте следующую строку:

```
127.0.0.1 uXXXXXX-lab1.local
```

(вместо XXXXXX укажите свой номер студенческого билета)

Сохраните изменения (в nano - нажатием Ctrl+O, далее Ctrl+X) и проверьте настройки, выполнив в терминале:

```
ping uXXXXXX-lab1.local
```

Если все настройки корректны, в выводе команды будет отображен IP-адрес, указанный в файле hosts.

## Настройка nginx и php-fpm

### Сервис nginx

Проверьте, что сервис nginx уже запущен:

```
sudo service nginx status
```

```
miet@ubuntu: ~/apps
● nginx.service - A high performance web server and a reverse proxy server
   Loaded: loaded (/lib/systemd/system/nginx.service; enabled; vendor preset: enabled)
   Active: active (running) since Fri 2021-12-10 11:31:19 MSK; 1h 25min ago
     Docs: man:nginx(8)
   Main PID: 1183 (nginx)
    Tasks: 17 (limit: 4599)
   Memory: 16.9M
   CGroup: /system.slice/nginx.service
           └─1183 nginx: master process /usr/sbin/nginx -g daemon on; master_process on;
             └─1184 nginx: worker process
               └─1185 nginx: worker process
                 └─1188 nginx: worker process
                   └─1189 nginx: worker process
                     └─1190 nginx: worker process
                       └─1191 nginx: worker process
                         └─1192 nginx: worker process
                           └─1193 nginx: worker process
                             └─1194 nginx: worker process
                               └─1195 nginx: worker process
                                 └─1197 nginx: worker process
                                   └─1198 nginx: worker process
                                     └─1199 nginx: worker process
                                       └─1201 nginx: worker process
                                         └─1202 nginx: worker process
                                           └─1203 nginx: worker process

дек 10 11:31:10 ubuntu systemd[1]: Starting A high performance web server and a reverse proxy server...
дек 10 11:31:19 ubuntu systemd[1]: Started A high performance web server and a reverse proxy server.
~
miet@ubuntu:~/apps$
```

Запуск сервиса nginx

```
sudo service nginx start
```

Остановить сервис nginx

```
sudo service nginx stop
```

Перезапуск сервиса nginx

```
sudo service nginx restart
```

Проверка конфигурации nginx

```
sudo nginx -t
```

Конфигурация nginx

Пример конфигурации

<https://www.nginx.com/resources/wiki/start/topics/examples/full/>

*Разбор основных директив см. в лекции №1.*

Создайте директорию /home/miet/apps/uXXXXXX-lab1 (где **XXXXXX** – номер студенческого билета, например, /home/miet/apps/u123456-lab1).

Создайте файл /home/miet/apps/uXXXXXX-lab1/**hello.php**

```
<?php
    echo "Hello from PHP";
```

Создайте файл-конфиг /etc/nginx/sites-available/uXXXXXX-lab1.local

Заполните директивы:

```
server {
    listen 80;
    server_name uXXXXXX-lab1.local;
    root /home/miet/apps/uXXXXXX-lab1;
}
```

Подключение php-fpm

Добавьте следующую директиву (внутри server)

```
location ~ \.php$ {
    fastcgi_pass unix:/var/run/php/php8.0-fpm.sock;
    fastcgi_param SCRIPT_FILENAME $realpath_root$fastcgi_script_name;
    include fastcgi_params;
}
```

Для того, чтобы виртуальный хост заработал необходимо создать символическую ссылку на этот файл и положить его в каталог sites-enabled, которые Nginx считывает при запуске. Создать ссылку можно следующей командой:

```
sudo ln -s /etc/nginx/sites-available/uXXXXXX-lab1.local /etc/nginx/sites-enabled/
```

Если внутри директории `/etc/nginx/sites-enabled/` есть конфиг по-умолчанию с названием `default` - удалите его (`cd /etc/nginx/sites-enabled/ && sudo rm -f example`)

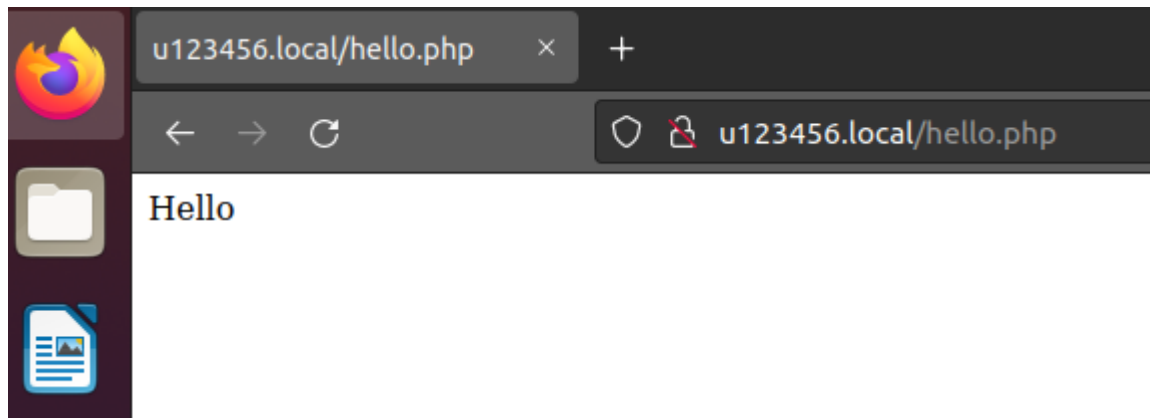
Проверьте конфигурацию nginx (`sudo nginx -t`) и перезагрузите nginx.

Проверьте что php-fpm запущен (`sudo service php8.0-fpm status`).

Если нет - запустите (`sudo service php8.0-fpm start`)

Проверьте, что скрипт выполняется по адресу

`http://uXXXXXX-lab1.local/hello.php`



Если страница не открывается - посмотрите логи ошибок в `sudo tail /var/log/nginx/error.log` и `sudo tail /var/log/nginx/access.log`

## Основы PHP

### PSR стандарты

#### [PSR-1 Basic Coding Standard](#)

Регулирует основные стандарты, главная идея которых – если все разработчики используют одни стандарты, то перенос кода можно производить без всяких проблем.

### [PSR-3 Logger Interface](#)

Основная цель данного интерфейса – простым и универсальным способом стандартизировать реализацию логирования. К данному интерфейсу прилагается спецификация, которая описывает в каких случаях какой из методов рекомендуется использовать.

### [PSR-4 Autoloading Standard](#)

В этом PSR описывается спецификация для классов автоматической загрузки из путей к файлам. В этом PSR также описывается, где размещать файлы, которые будут загружаться автоматически в соответствии со спецификацией.

### [PSR-6 Caching Interface](#)

Кэширование широко используется для повышения производительности любого проекта.

Кэширование также является одной из наиболее распространенных функций многих CMS, фреймворков и библиотек.

Это привело к ситуации, когда многие библиотеки реализуют свои собственные системы кэширования с различными уровнями функциональности. Эти различия заставляют разработчиков изучать несколько систем, которые могут предоставлять или не предоставлять необходимую им функциональность. Кроме того, разработчики кэширующих библиотек сами сталкиваются с выбором между поддержкой только ограниченного числа платформ или созданием большого количества классов адаптеров.

Чтобы решить эти проблемы был принят общий стандарт для библиотек реализующих кэширование, он включает в себя несколько интерфейсов.

### [PSR-7 HTTP Message Interface](#)

В этом PSR описываются общие интерфейсы для представления HTTP-сообщений.



### [PSR-11 Container Interface](#)

Основная цель стандартизировать, как фреймворки и библиотеки будут использовать (DIC) контейнер для доступа к объектам и параметрам. Для этого был описан ContainerInterface. Спецификация PSR-11 не описывает то, как необходимо регистрировать зависимости в проекте, однако дает четкую рекомендацию как делать не нужно.

### [PSR-12 Extended Coding Style Guide](#)

Эта спецификация расширяет и заменяет устаревшее руководство по стилю кодирования (PSR-2), и требует соблюдения базового стандарта кодирования (PSR-1).

### [PSR-13 Hypermedia Links](#)

Не самый популярный стандарт, который предоставляет несколько интерфейсов, чтобы унифицировать общий формат hypermedia ссылок. В качестве примера, можно рассмотреть использование hypermedia ссылок в контексте HTML и в различных форматах API.

### [PSR-14 Event Dispatcher](#)

Целью этого PSR является создание общего механизма для диспетчеризации событий, чтобы библиотеки и компоненты могли свободно использоваться в различных приложениях и средах. Диспетчеризация событий – это распространенный и хорошо протестированный механизм, позволяющий разработчикам легко и последовательно расширять логику приложения.

### [PSR-15 HTTP Handlers](#)

В этом PSR описывается стандарт обработчиков запросов и middleware.

## [PSR-16 Simple Cache](#)

Обратите внимание на [PSR-6](#), это действительно «мощная» спецификация для реализации системы кэширования, однако в большинстве проектов такая реализация может оказаться избыточной. Поэтому был принят PSR-16. Этот более простой подход направлен на создание стандартизированного оптимизированного интерфейса для общих случаев.

## [PSR-17 HTTP Factories](#)

PSR-17 описывает общий стандарт для фабрик, которые создают HTTP-объекты, совместимые с PSR-7.

PSR-7 не содержит рекомендации о том, как создавать HTTP-объекты. Это может приводить к трудностям при необходимости их создания внутри компонентов, которые не привязаны к конкретной реализации PSR-7.

## [PSR-18 HTTP Client](#)

PSR-18 описывает общие интерфейсы для отправки PSR-7 HTTP-запросов и получения HTTP-ответов. Это может сделать библиотеки более пригодными для повторного использования, так как уменьшает количество зависимостей и снижает вероятность конфликтов версий.

Также в спецификации указано, что HTTP-клиенты могут быть заменены согласно принципу подстановки Лисков. Это означает, что все клиенты ДОЛЖНЫ вести себя одинаково при отправке запроса.

## PHP Coding Standards Fixer

[PHP-CS-Fixer](#) – инструмент, который помогает исправить код в соответствии со стандартами. PHP-CS-Fixer позволяет не только обнаруживать проблемы, но и устраняет их за вас. Если вам нужно

применить стили кода, которые не поддерживаются инструментом, вы можете создавать свои собственные правила.

*Документация и инструкция по установке:*

<https://github.com/FriendsOfPHP/PHP-CS-Fixer#documentation>

## Логирование сообщений

[Monolog](#) – библиотека реализующая интерфейс PSR-3.

*Основные свойства Monolog:*

**Logger** – это объект, который мы используем для записи логов. Сам логгер не занимается записью, но управляет хендлерами. Может быть создано любое количество.

**Handler** – объект, который непосредственно обрабатывает данные. Вы можете добавить сколько угодно хендлеров в логгер. Все они будут вызываться по очереди, независимо от того смог данный хендлер обработать ошибку или нет.

**Processor** – любая вызываемая сущность (callable). Может быть несколько. Могут быть назначены как глобально, так и установлены для хендлера. Сначала запускаются глобальные процессоры. Основная задача процессора добавить дополнительные данные в лог (например IP, с которого был коннект, значение глобальных переменных, информация на какой git ветке сейчас находится код и так далее).

**Formatter** – преобразует вывод сообщения перед записью. Может быть только 1 на хендлер. Нужен для изменения форматирования тела сообщения, например для преобразования текста в html или json.

**Channel** – имя логгера. Оно будет написано при записи лога. Так как 1 хендлер может быть применен в 2ух разных логерах (будет писать логи в 1 и тот же файл) это позволит определить откуда пришла ошибка.

**Level** – уровень ошибки. Данный параметр у хендлера означает минимальный уровень ошибки, который будет им обрабатываться.

**Bubble** – всплытие сообщения. После того, как хендлер обработал сообщение, то логгер передает сообщение следующему хендлеру. Этот процесс можно остановить с помощью свойства bubble. Если у хендлера значение этого свойства false (по умолчанию всегда true), то после того как этот хендлер выполнит свою работу (смог обработать данную ошибку), другие хендлеры уже не запустятся.

**Sort Order** – порядок выполнения. Последний добавленный хендлер всегда запускается самым первым. Именно эта особенность позволяет внедрять механизм полного отключения логгера (через bubbling false). Хендлеры добавленные через конструктор идут в порядке указанном в конструкторе.

Документация, инструкция по установке и примеры:

<https://github.com/Seldaek/monolog>

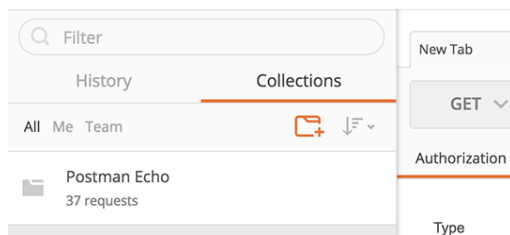
## Основы Postman

Postman предназначен для проверки запросов с клиента на сервер и получения ответа от бэкенда.

*Основные объекты:*

**Коллекция** – отправная точка для нового API.

Коллекция объединяет в себе все связанные запросы. Обычно API описывается в одной коллекции, но если вы желаете, то нет никаких ограничений сделать по-другому. Коллекция может иметь свои скрипты и переменные, которые мы рассмотрим позже.



**Папка** – используется для объединения запросов в одну группу внутри коллекции. Папка, как и коллекция может иметь свои скрипты, но не переменные.

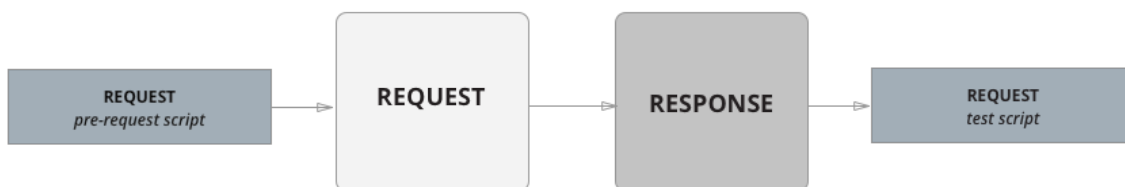
**Запрос** – основная составляющая коллекции.

Запрос создается в конструкторе. Postman умеет выполнять запросы с помощью всех стандартных HTTP методов. У запроса есть свои скрипты. Обратите внимание на вкладки «Pre-request Script» и «Tests». Они позволяют добавить скрипты перед выполнением запроса и после.

**Скрипты.** «Postman Sandbox» это среда исполнения JavaScript доступная при написании «Pre-request Script» и «Tests» скриптов.

«Pre-request Script» используется для проведения необходимых операций перед запросом, например, можно сделать запрос к другой системе и использовать результат его выполнения в основном запросе.

«Tests» используется для написания тестов, проверки результатов, и при необходимости их сохранения в переменные.

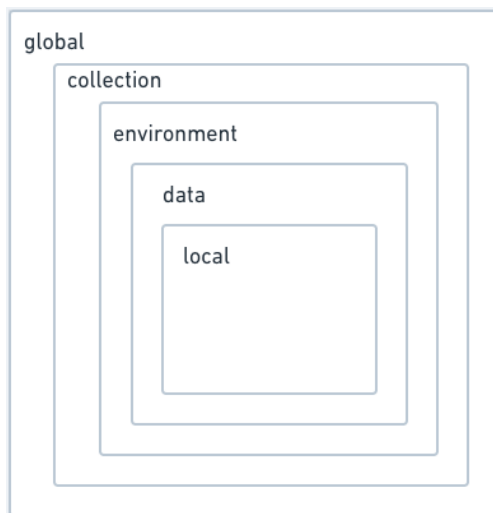


## Установка и получение переменных

```
// глобальные переменные
pm.globals.set("key", "value");
pm.globals.get("key");

// переменные окружения
pm.environment.set("key", "value");
pm.environment.get("key");

// локальные переменные
pm.variables.set("key", "value");
pm.variables.get("key"); // если нет локальной, будет искать на уровне
                           // выше
```



Получить значения в конструкторе  
{variable\_key}

Получить значения из скриптов

```
// получить глобальную переменную
pm.globals.get("variable_key");

// получить переменную из окружения
pm.environment.get("variable_key");

// получить переменную из любого
// пространства согласно приоритету
pm.variables.get("variable_key");
```

# Практическая часть

## Общие задания

### Задание 1.1

Самостоятельно изучите следующие git-команды:

`git init`

`git pull`

`git push`

`git fetch`

`git add`

`git commit`

`git status`

`git clone`

`git checkout`

`git checkout -b`

`git checkout -- file`

`git reset (--hard, --soft)`

`git branch`

`git log`

`git cherry-pick`

`git merge`

`git config user.name "John Doe"`

`git config user.email "foo@example.ru",`

(понять что делает флаг `--global`)

### Задание 1.2

Создайте новый (*пустой*) проект в PhpStorm / VSCode, а также создайте публичный репозиторий на [gitlab.com](https://gitlab.com)

Попробуйте запустить команды из задания 1.1.



### Задание 1.3

Настройте виртуальный хост `uXXXXXXX-lab1.local`

(вместо XXXXXX укажите свой номер студенческого билета)

*\* все последующие задания должны работать с хостом `uXXXXXXX-lab1.local`*

### Задание 1.4

С помощью composer установите библиотеку [symfony/validator](https://symfony.com/doc/4.4/components/validator.html)

Сгенерируйте файл для автозагрузки (composer dump-autoload)

Подключите autoload.php в свой проект.

Продemonстрируйте работу валидатора (в своём проекте) на примерах из [документации](#).

## Задания по вариантам

### Вариант №1

#### Задание 2.1

Создайте класс User, который в конструкторе принимает id пользователя, имя, email и пароль. Для каждого свойства создайте собственные правила валидации. Создайте несколько пользователей и продемонстрируйте работу валидатора. Добавьте метод, который возвращает дату и время создания текущего объекта (*пользователя*).

#### Задание 2.2

Создайте класс Comment, который в конструкторе принимает объект класса User и текст сообщения. Создайте несколько комментариев, поместите их в массив/коллекцию. Пройдитесь по всем комментариям и выведите только те комментарии, пользователи которых были созданы после \$datetime (объект класса [DateTime](#), задается пользователем).

## Вариант №2

### Задание 2.1

Создайте класс `Employee`, который в конструкторе принимает `id` сотрудника, имя, размер зарплаты, дату принятия на работу. Добавьте метод, который возвращает текущий опыт работы сотрудника (количество полных лет). Для каждого свойства создайте собственные правила валидации. Создайте несколько пользователей и продемонстрируйте работу валидатора.

### Задание 2.2

Создайте класс `Department`, который в конструкторе принимает массив объектов класса `Employee` (*сотрудников*) и название. Добавьте метод, который возвращает суммарный размер зарплаты сотрудников. Создайте несколько объектов класса `Department`, поместите их в массив/коллекцию. Выведите отделы с наименьшим и с наибольшим размером суммарной зарплаты. Если несколько отделов имеют одинаковый размер суммарной зарплаты, выведите тот отдел, который имеет большее количество сотрудников (если и этот показатель равный – выведите все подобные отделы).

## Список литературы

### PHP

1. <https://code-basics.ru/languages/php>
2. <https://daylerees.com/php-pandas/>
3. <https://phptherightway.com/>
4. <http://php.net/manual/ru/>
5. <https://getcomposer.org/>

### Git

1. <https://git-scm.com/book/ru/v1>
2. <https://learngitbranching.js.org>