1. PODIZANJE HYPERV

- https://ubuntu.com/download/desktop
- Virtual switch manager – definirti switch
- **Prikazivanje IP adrese**
  - **ip addr show-**
  - ip a
- **Promjena root PWD:**
  - sudo passwd root
- **Promjena vlasništva na direktoriju**
  - sudo chown -R $USER:$USER /var/www/html/ostechnix1.lan/public_html
- **Install apache on Linux**
  - Sudo apt update
  - Sudo apt install apache2
- **Konfiguracija Appache**
- **Promjena default www**
  - sudo nano /etc/apache2/sites-enabled/000-default.conf
- **Restart Appache**
  - sudo systemctl restart apache2
- **Mjesta gdje možeš pronaći .htaccess na Apache2:**
  - .htaccess datoteka se najčešće nalazi u root direktoriju vaše web stranice. Ovisno o konfiguraciji vašeg servera, to će biti jedan od sljedećih direktorija:
    - /var/www/html/ (Ako koristite osnovnu instalaciju)
    - /var/www/yourdomain.com/ (Ako imate nekoliko web stranica)
    - /home/username/public_html/ (Ako koristite shared hosting)

Na primjer, ako vaša stranica koristi direktorij /var/www/html/, .htaccess bi se trebala nalaziti tamo.

- **Konfiguracija MySql**
  - sudo apt install mysql-server

```
1. sudo mysql
2. ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql_native_password BY
   'password';
3. Mysql exit;
```

  - sudo mysql_secure_installation (promjena pwd za root)
  - mysql -u root -p (pristup bazi s root korisnikom)

- **Instalacija PHP**
  - sudo apt install php libapache2-mod-php php-mysql
  - php -v
  - PHP ini:
  - /etc/php/7.4/apache2/php.ini

- **Instalacija Composer-a:**
    - **Korak 1: Preuzmi Composer installer**

      https://www.digitalocean.com/community/tutorials/how-to-install-and-use-composer-on-ubuntu-18-04

    - cd ~
    - php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
    - **Korak 2: Verificiraj hash**
    - cd ~
    - curl -sS https://getcomposer.org/installer -o /tmp/composer-setup.php HASH=`curl -sS https://composer.github.io/installer.sig`
    - echo $HASH
    - php -r "if (hash_file('SHA384', '/tmp/composer-setup.php') === '$HASH') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"

    - **Instalacja composer-a**
    - sudo php /tmp/composer-setup.php --install-dir=/usr/local/bin --filename=composer

PRIMJER:

Kreiranje mape za projekat

Sada, kreiraj traženu mapu **ComposerExercise** u direktoriju /var/www/html/BackendDeveloper.

Korak 1:Promjeni vlasnika na BackendDeveloper

sudo chown -R $USER:$USER /var/www/html/BackendDeveloper

Korak 2: Kreiraj mapu

sudo mkdir -p /var/www/html/BackendDeveloper/ComposerExercise

Korak 2: Prebaci se u novokreiranu mapu

cd /var/www/html/BackendDeveloper/ComposerExercise

3. Instaliraj paket sa Packagist-a

Za primjer, instalirat ćemo paket guzzlehttp/guzzle (popularni HTTP client paket).

Korak 1: Pokreni Composer inicijalizaciju u mapi

composer init

Ovdje ćeš unijeti osnovne informacije o projektu (ime, opis itd.).

Korak 2: Instaliraj željeni paket

composer require guzzlehttp/guzzle

4. Provjeri instalaciju

Nakon instalacije paketa, sve zavisne datoteke bit će preuzete u mapu vendor/, a composer.json i composer.lock datoteke će biti kreirane. Provjeri sadržaj mape:

ls -l

- strukturu projekta s composer.json, composer.lock i vendor/ mapa.

2. **Git:**

- kreirati direktorij
- pokrenuti git init – definira se glavna grana
- preimenovati master u main – **git branch -M main**
- dodati neku datoteku u master granu
- pokrenuti git add .
- pokrenuti git commit -m „App init“
- git branch – pokazuje grane

Na main branchu se drži kod koji je u produkciji, na njemu se ne radi

Git branch naredba pokazuje u kojem smo branchu – pokazano sa *****

- **Prebacivanje na drugu granu**
  - Git checkout „ime grane“
- **Stvaranje nove grane i automatski prebacivanje**
  - Git checkout -b „naziv grane“

Nazivi za grane: https://gist.github.com/qoomon/5dfcdf8eec66a051ecd85625518cfd13

- **Brisanje brancheva**
  - Pozicioniranje u branch koji nećemo brisati
  - Moramo se pozicionirati u main - **git checkout main**
  - Brisanje - git branch -d „ime brancha“
  - Ako ima neprenesenih promjena force delete **git branch -D „ime brancha“**
- **Spajanje promjena n jednom branchu u drugi**
  - Pozicionirani u branch u koji ćemo mergati
  - **Git merge featureA**
  - **prebaci i commit**

- **Ako radimo na istoj datoteci u dvije različite grane, a želimo promjene comitati u treću**

- **Pokrenemo git add .**
- **Pokrenemo git commit -m „opis priomjene“ na svakoj grani**
- **Prebacimo se u granu u kojoj želimo spojiti promjene:**
  - **Pokrenemo git merge „naziv grane 1“**
  - **Pokrenemo git merge „naziv grane 2“**

U VS studio odaberemo što želimo napraviti s promjenama i nakon toga pokrenemo

Git add . i git commit – m „Fix ....“

Git log - popis svih commiteva

git checkout <commit-hash> - vraćanje na neki od commiteva

Udaljeni repozitorij:

Github /dashboard

Prijenos https protokol – uvijek pita za lozinku kad želimo nešto prenijeti na repozitoriji, traži svaki puta prijavu

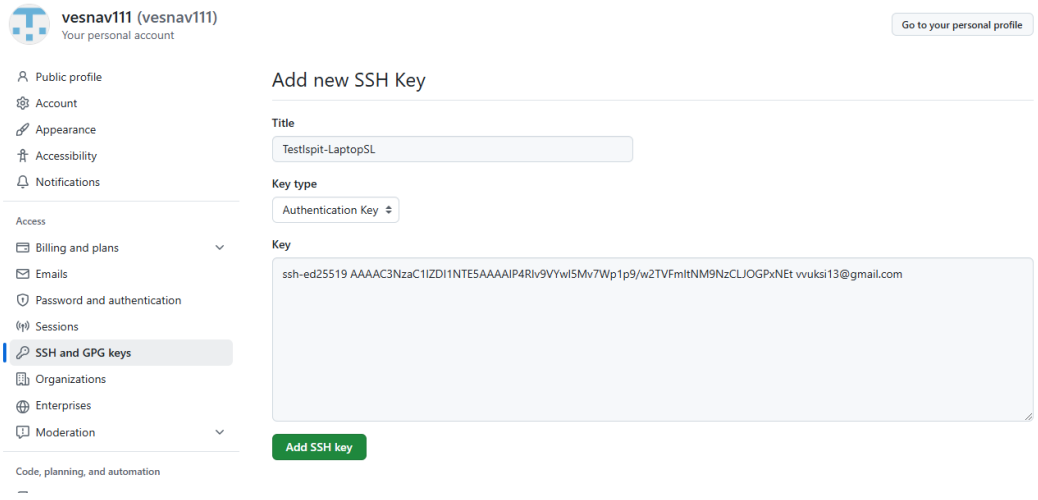
Ssh protokol – uspostavlja sigurni vezu i mehanizam s kojim osiguravamo da se naše računalo predstavi git hubu bez unošenja lozinke

- **Ssh ključevi**
    - Prenese javni ključ u git hub servis – predstavljamo se s tim ključem

https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

Pokrenemo: ssh-keygen -t ed25519 -C "your_email@example.com"

- odaberemo naziv ključa
- generira dvije datoteke : javnu i privatnu (javni i privatni ključ)
- javni ključ se daje servisima i na taj način komuniciramo, privatni ne ide nigdje
- kopiramo public ključ u notepadu i odemo na git hub



Kreirati udaljeni repozitorij

- Gumb novi repozitorij and
- Prebaciti s https u ssh
- git remote add origin git@github.com:vesnav111/git-branches.git
- git remote add origin git@github.com:tkescec-algebra/laravel-algebra-blog.git
- git remote -v – provjera statusa
- git branch -a - pokazuje sve grane i online

Povezivanje s dev grane na main granu online
Zadnji korak odabrati „Megre pull request“ - odabrati „rebase and merge“ – da se branchevi svedu na istu razinu
Pull request – smisao ima jedino kad se radi u timu, šalje se na odobrenje, team lead prihvaća je li kod u redu

Semantički sustav verzioniziranja:

Semantički sustav verzioniranja (eng. Semantic Versioning, često skraćeno kao SemVer) je standard za označavanje verzija softvera koji pomaže programerima i korisnicima razumjeti

promjene u softveru na temelju brojeva verzija. Koristi format MAJOR.MINOR.PATCH (npr. 1.4.2), gdje svaki broj ima jasno definirano značenje:
Komponente verzije:
MAJOR (glavni broj):
Mijenja se kada se uvedu promjene koje nisu kompatibilne s prethodnim verzijama (tzv. "breaking changes").
Npr. prelazak s 1.x.x na 2.0.0 označava da su određene funkcionalnosti promijenjene ili uklonjene te možda neće raditi s kodom koji koristi stariju verziju.
MINOR (manji broj):
Mijenja se kada se dodaju nove funkcionalnosti koje su kompatibilne s postojećom verzijom.
Npr. prelazak s 1.3.0 na 1.4.0 označava dodavanje novih značajki bez utjecaja na postojeći kod.
PATCH (zakrpa):
Mijenja se kada se isprave greške ili uvedu manje izmjene koje ne mijenjaju funkcionalnost (kompatibilne su).
Npr. prelazak s 1.3.2 na 1.3.3 označava ispravak buga bez utjecaja na ponašanje sustava.
Pravila semantičkog verzioniranja:
Verzija započinje s 0.1.0 tijekom razvoja (prije stabilnog izdanja). Verzija 0.x.y se smatra nestabilnom i može se često mijenjati. Kada softver postane stabilan, prelazi na 1.0.0.
Brojevi se povećavaju na temelju vrsta promjena:
"Breaking changes" -> povećava MAJOR.
Dodavanje kompatibilnih značajki -> povećava MINOR.
Ispravci grešaka -> povećava PATCH.

- **KLONIRANJE REPOZITORIJA**
  - Kreira se repozitorij
  - Nakon toga odabere se suradnik
  - Suradnik dobije poveznicu
  - Kopira kod – lokalno računalo + ssh
  - Lokalno se s git bash pozicioniramo gdje želimo i napravimo clone
  - **Git clone + kopiramo kod**
  - Kreiramo svoju granu
  - Kad smo gotovi napravimo **Git push -u origin „naziv grane“**
  - Dolazi na odobrenje team leadu

Knjiga

https://tkrajina.github.io/uvod-u-git/git.pdf

Dokumentacija

https://git-scm.com/docs

Kako generairati SSH ključ

https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent

2. OSNOVE PHP

https://www.php.net/

a) Php direktive

Bitno je gdje se napravi include, na kojem mjestu u datoteci**Ako include ne pronađe datoteku dobije se samo upozorenje**

Include - Naredba uključi uključuje i procjenjuje navedenu datoteku.

Include once – Naredba include_once uključuje i procjenjuje navedenu datoteku tijekom izvođenja skripte. Ovo je ponašanje slično naredbi uključivanja, s jedinom razlikom što ako je kod iz datoteke već uključen, neće biti ponovo uključen. Kao što naziv sugerira, bit će uključen samo jednom.

include_once se može koristiti u slučajevima kada ista datoteka može biti uključena i procijenjena više puta tijekom određenog izvođenja skripte, tako da u ovom slučaju može pomoći u izbjegavanju problema kao što su redefinicije funkcija, preraspodjele vrijednosti varijable itd.

uključi_jednomrequire

Reqiuire - require je identičan uključivanju osim što će u slučaju neuspjeha proizvesti fatalnu pogrešku razine E_COMPILE_ERROR. Drugim riječima, zaustavit će skriptu, dok uključi samo emitira upozorenje (E_WARNING) koje omogućuje nastavak skripte.

Require_once - Naredba require_once je identična kao require osim što će PHP provjeriti je li datoteka već uključena, i ako je, neće je ponovno uključiti (zahtijevati).

error_reporting (0) – u php.ini ugasi greške u php i worning, ma developmentu mora biti upaljeno

b) Varijable

Varijabla – u nju se sprema podatak kojeg kasnije koristimo i ona je varijabilna, vrijednosti u varijabli se mijenjanju

Konstanta - konstanta sadrži vrijednost isto kao i varijabla, ali za razliku od varijable, kojoj se vrijednost može mijenjati tijekom izvođenja programa, vrijednost konstante se ne mijenja

```
define('PI', 3.14);
```

String varijanle definiraju se pomoću dvostrukih ili jednostrukih navodnika

Prednost dvostrukih navodnika je da unutar vrijednosti može koristiti specijalne znakove (npr. \za novi red) i za konaktenaciju ne mora koristiti točku nego samo navede ime varijable

Reference – služe kako bi jednu varijablu referencirali na drugu. Promjenom varijable na koju je referencirano mijenja se i vrijednost referencirane varijable. Kod referenciranja stavlja se **&**ispred reference

Varijable nije potrebno unaprijed deklarirati

Varijabla bez vrijednosti – deklaracija, kad joj se pridruži vrijednost – inicijalizacija

Echo je funkcija koja može ispisati jedan ili više stringova i ne vraća ništa

Naredba **print** ispisuje samo jedan string i vraća uvijek 1.

Ako pomoću funkcije Echo želimo ispisati nešto što nije string npr. boolean događa se **Type juggling** i ne ispisuje se true, nego je taj true pretvoren u **1, false ne ispisuje (u svijetu stringa je to prazan string) -implicitna konverzija**

Type casting – mi forsiramo promjenu vrste podatka u varijabli **echo (int) $varijabla – explicitna konverzija**

Primjeri ispisa:

- Kad se zapisuje u datoteku novi red **\n**
- U browseru **
** (tag)
- **** - escape znak

Konkatenacija = $ime . „ „ .$prezime

Interpolacija = „$ime $ prezime“ (ne radi unutar jednostrukih navodnih znakova)

Null – nema vijednost, ispravnije je odmah inicijalizirati vrijednost (ne samo $var)

isSet – vraća je li varijabla postavljena

c) Operetori

$++ (postfix operator)

$b=20

$c=$b++ ($c=20, $b=21)

++$ (prefix operator)

$b=20

$c=++$b ($c=21, $b=21) - ++ ima veći prioritet od = (sučelila su se dva operatora i gleda se prioritet koji je jači)

- **Refernca**

Kad jednoj varijabli dodijelimo vrijednost druge varijable **(&)** – ispred varijable na desnoj strani, svaki puta kada se u jednoj varijabli promijeni vrijednost promijeni se vrijednost i u drugoj varijabli

(vrijedi i u suprotnom smjeru)

Smisao reference je da se mijenjaju vrijednosti globalnih varijabli unutar funkcija

- **Stringovi**

Problem s encodingom, funkcije koje počinju sa str rade jedino sa ASCII znakovima (ne rade dobro s našim znakovima npr.

$ime=“SAŠA“

strtoupper ($ime) vrati SAšA (svali znak je jedan byte, a znak Š ima 2 znaka), moramo koristiti funkciju multibyte **mb_strtoupper,** ima parametar encoding (utf-8)

crosssidescripting - sigurnosni problem – podvali se skripta, iz PHP se pošalje u JS cod, browser izvrši taj kod kad dobije tagove <script></script>

kad se gleda input od strane korisnika nikada se ne upisuju drrektno, postoje funkcije koje čiste taj string, **htmlspecialchar() ili htmlentities()**

– mijenja problematične znakove u specijalne znakove, skripta se ne izvršava nego ispisuje

echo htmlspecialchars("<script>alert(\" DGSDGSDGSDG \")</script>");

3. NIZOVI (Array)

Liste – poredani skup elementa

Mapa – sastoji se od ključa i vrijednosti, nebitan je redoslijed

Niz se inicijalizira sa **[]** ili **Array** funkcijom

Višedimenzionalni nizovi – moguće – ugniježđeno

Kad u jednostavni niz želimo **umetnuti** element **na kraj** pišemo:

$imeniza []='Vrijednost', treba umetati preporuka uvijek na kraj

Mapa

$imeključa['naziv svojstva'] = „vrijednost“

a) Funkcije za provjeru Niza:

Empty – provjera je li niz prazan

Isset – provjerava se je li neki ključ uopće postavljen

Unset – ukloniti elemente iz niza

Unset je ok kod mapa, ako uklanjamo element iz lista kad maknemo vrijednost, unset reinedksira listu

Traženje elemenata – ugrađene funkcije (in_array, array_serch)

- **Array_search** – vraća indeks pod kojim je zapisana ta vrijednost, ako ne nađe javlja false
- **In_array** – vraća true ili false

Sortiranje Array-a – ugrađene funkcije – **mergesort** – postoje algoritmi, Potrebno je optimizirati sortiranje

Prazan Array =false;

Dodavanje elementa:

- **Array_push** – dodaje element na kraj arrey-a
- **Micanje prvog elementa - Unset ($fruits[0]); -** indexsi se zadrže, samo se makne element
- **Array_shift –** makne prvi element i preindexsira
- **Array_merge** – spaja dva niza ili
  $spojeni niz=[...$prvi, ...$drugi] – (... - operator dekompozicije – rastavi niz na članove

array_filter(array **$array, ?**callable **$callback =** null**,** int **$mode = 0):** array – ide element po element i šalje callback funkciji parametar, odgovor iz Callback funkcijie mora biti true ili false (npr vrati samo brojeve koji su veći od 5) – piše u novi array, funkcija je anonimna i ne može se pozvati, živi samo kad se izvršava array_filter, novi array ostaje indexsacija

b) PHP kontrolne strukture
Usmjeravaju tijek izvršenja koda

- **Petlje i Uvjetovane**
- **Petlje**

Što su petlje?
Petlja izvršava slijed naredbi mnogo puta dok navedeni uvjet ne postane lažan. A petlja sastoji se od dva dijela, tijela petlje i kontrolne naredbe. Kontrolna izjava je kombinacija nekih uvjeta koji usmjeravaju tijelo petlje da se izvrši sve dok navedeni uvjet ne postane lažan. Svrha petlje je ponavljanje istog koda više puta.
Što je while petlja?
To je ulazno kontrolirana petlja. U while petlji, uvjet se procjenjuje prije obrade tijela petlje. Ako je uvjet istinit, tada i samo tada se izvršava tijelo petlje.
Nakon što se tijelo petlje izvrši, kontrola se ponovno vraća na početak, a uvjet se provjerava. Ako je istinit, isti se proces izvršava sve dok uvjet ne postane lažan. Jednom kada uvjet postane lažan, kontrola izlazi iz petlje.
U while petlji, ako uvjet nije istinit, tada se tijelo petlje neće izvršiti, niti jednom.
Što je Do-While petlja?
Do-while petlja slična je while petlji osim što se uvjet uvijek izvršava nakon tijela petlje. Također se naziva petlja kontrolirana izlazom.
U do-while petlji, tijelo petlje uvijek se izvršava barem jednom. Nakon što je tijelo izvršeno, onda provjerava stanje. Ako je uvjet istinit, tada će ponovno izvršiti tijelo petlje. U suprotnom, kontrola se prenosi izvan petlje.

While ili do while se koristi – kad se ne zna broj ponavljanja – dok s ne ispuni uvijet

For petlja koristi se kad znamo broj ponavljanja i sastoji se od:
-iterator
-uvijet koji uključuje iterator
-promjena iteratora

Foreach – koristi se u radu s nizovima
- Radi sa elementima koji imaju implementirano sučelje Countable
- Iterator unutar petlje
- Array ima svojstvo prebrojivosti
- Ne trebaju se definirati iteratori ni uvjeti, prebrojiv element uvjetuje do kada petlja radi

For petlja:

Break – zaustavlja petlju

Continue - preskače korak prekida trenutnu iteraciju i nastavlja izvođenje petlje

Return – ponaša se kao bake, vraća rezultat koji smo dobili izvršavanjem neke funkcije

- **Uvjetovane strukture**

  If - neovisni if
  If/ else – else se izvrši ako if uvije nije zadovoljen
  If / else if – međusobno su vezani, ne pokreću se sve dok prethodni uvijet nije zadovoljen
  Switch case – obavezan break; nakon što je uvijet zadovoljen

c) FUNKCIJE

Overload – svojstvo funkcije da imamo funkciju s različitim brojem parametra kod proceduralnog programiranja

Pravilo za imena funkcije **Camel case** naziv funkcije – prva riječ počinje malim slovom ostale sve velikim

Funkcija može ili ne mora vratiti vrijednost, ako ne vraća vrijednost to se naglašava sa **:void** kod definiranja funkcije

Plimorfizam – jedna metoda u različitim objektima radi različite stvari

Stroga tipizacija – određuje koji tip podatka funkcija mora vratiti, direktiva koja uključuje strogu tipizaciju ***Declare(strict_types=1); - koji tip podatka može ući u funkciju i što funkcija može vratiti***

Kod definicije funkcije :string

Parametar je varijabla koja se nalazi u zagradi funkcije.

Argument je nekakav izraz koji šaljemo u parametar funkcije.

Postoji mogućnost da se definiraju opcionalni parametri to znači da već ti parametri imaju već definiranu vrijednost

Imenovani argumenti – neovisno o redoslijedu

Pozicijski argumenti prvi argument na prvu poziciju, drugi argument na drugu ...

Varijable – imaju globalni i lokalni scope

Sve globalne varijable pohranjuje u jednu superglobalnu varijablu $GLOBALS

-imenski prostori

$GLOBALS – bilo gdje u kodu može se doći do varijable putem $Globals i naziva varijable – roditeljski array i sve varijable koje se kreiraju idu u taj array – ako već moramo raditi sa superglobalnim varijablama

Izbjegava se to i definiraju se **lokalne varijable** i varijabla živi samo unutar te funkcije – o tome brinu garbbage collectori, ako se želi unutar funkcije doći do superglobalne varijable piše se

$GLOBALS['imevarijble']

Primjer custom Callble funkcije:

```
function array_f(array $a, callable $callback): array{
  $newArray=[];
  foreach ($a as $value){
    if ($callback($value)){
      $newArray[]=$value;
    }
  }
return $newArray;
}
$a=array_f([5,10,15], function($value){
  return $value >5;
});
print_r ($a);
```

Ugrađena funkcija array_filter

```
$brojevi = [1,8,6,4,8,3,7,1,0,5];
$brojeviVeciOdPet = array_filter($brojevi, function($broj){
  return $broj > 5;
});
$reindeksiraniBrojevi = array_values($brojeviVeciOdPet);
print_r($reindeksiraniBrojevi);
```

Parametri:

1. **array $a**: Ovo je ulazni niz (array) koji funkcija obrađuje.
2. **callable $callback**: Funkcija koja se prosljeđuje kao argument (callback funkcija). Ova funkcija definira uvjet koji svaki element niza treba ispuniti kako bi bio uključen u rezultat.

Kako radi:

1. **$newArray = []**: Stvara se novi, prazan niz u koji će se dodavati elementi koji zadovoljavaju uvjet.
2. **foreach ($a as $value)**: Petlja prolazi kroz svaki element ulaznog niza $a.
3. **if ($callback($value))**:
    - Poziva se funkcija proslijeđena kao *callback* s trenutnom vrijednošću $value.

    - Ako funkcija vrati true, trenutna vrijednost se dodaje u $newArray.
4. **return $newArray**: Na kraju, funkcija vraća novi niz koji sadrži samo elemente koji su prošli uvjet.

Callback funkcija je u stvari funkcija u funkciji, proširenje funkcionalnosti glavne funkcije

Reference – parametar u funkciji je referenca &$imeparametra na neku drugu varijablu, šalje se argument koji je refernca na neku varijablu u programu

Rekurzija – funkcija koja poziva samu sebe

U memoriji se markiraju koraci, kako bi se po tim koracima znali vratiti (faktorijeli), koristi se u matemtici, fibobonacijevi niz, sort algoritmi (merge sort)

- base case
- korak rekurzije
- uvijet kada rekurzije prestaje, rekurzija radi na stack memoriji, dok petlje rade direktno na cpu, samo odrđeni djelovi petlje rade na stack-u
  stack memorija omogućava da markira stvari i onda se vraća

  Nedostatak rekurzije – kad potroši memoriju u problemu smo

  Pelja – obrađuje svi elementi iteracije, rekurzija vrti tako da na stek zapisuje sve dok ne dođe do baznog slučaja, kad dođe do baznog slučaja koristi markacije da izvuče rezultat

Static varijble – definira varijabliu kao memorijski prostor koji ostaje zapamćen između poziva funkcija, ne događa se ponovna inicijalizacija na nulu
Static ključna riječ ima mogućnost pamćenja između poziva funkcija

Funkcija kao varijabla – ime funkcije pridruži se varijabli i onda se varijabla poziva kao funkcija

Generatori – funkcija kojom možemo generirati različite stvari tko da ga zadržimo do iduće iteracije, generator omogućuje da funkciju pauziramo do iduće iteracije, funkcija im je u radu s datotekama, omogućuje da obrađujemo liniju po liniju, ada ne učitavamo cijelu datoteku.

Koristi **yeald** – znači vrati vrijednost, ali pauziraj funkciju do idućeg pokretanja, funkcija generator vraća objekt, sadrži pokazivače na kojem se property-u nalazimo

1. RAD S DATOTAKAMA

Tip podataka - resource

Bitna je putanja do datoteke

- Prvo kreiramo resorce – **fopen** vraća resorce, trebamo dobiti pointer na datoteku kako bi mogli čitati iz datoteke, omogućuje da čitamo znak po znak, fopen ima različite modove
  https://www.php.net/manual/en/function.fopen.php
  filesize, zna kad file završava
  fread – čita datoteku, ako je parametar fsize učita cijelu datoteku
  nakon što smo završili s datotekom moramo završiti s datotekom i osloboditi memoriju **fclose**

Ako ne može pročitati datoteku kod fopen dodati or die(„Ne mogu otvoriti datoteku $file name“)
Feof funkcija ispituje je li kraj dtoteke
Fgets – daje red iz dtoteke i prebaccuje pointer na novi red

File_get_contents() – vrati sadržaj datoteke kao string za razliku od file(), može vratiti da nije pronašao datoteku - **false**
Json_decode - očekuje string, a može dobiti i false, drugi element je **true** (vrati asocijativni array)
Stroga tipizacija donekle popravi problem

Spremanje u json datoteku: **File_put_contents**

Const BOOKS_FILE = „books.json“ – definira konstantu isto kao i define('konstanta', 3.14)

Json – java script object notation, nastao 1999

Ako u parametru upišemo **?string $title** - parametar je opcionalno string ili null, ali smo i dalje dužni poslati value i postavimo defult value na **null**

Zašto

Prazan string možemo tretirati kao da nema vrijednosti, ali prazan string je i dalje po tipu podatka string,

Foreach ($books as $book) – staje kad nađe na prvi element koji zadovoljava uvijet

Array_map – ne staje kod prvog uvijeta nego ide dalje do kraja arrey-a (npr dohvati će knjige nekog autora)

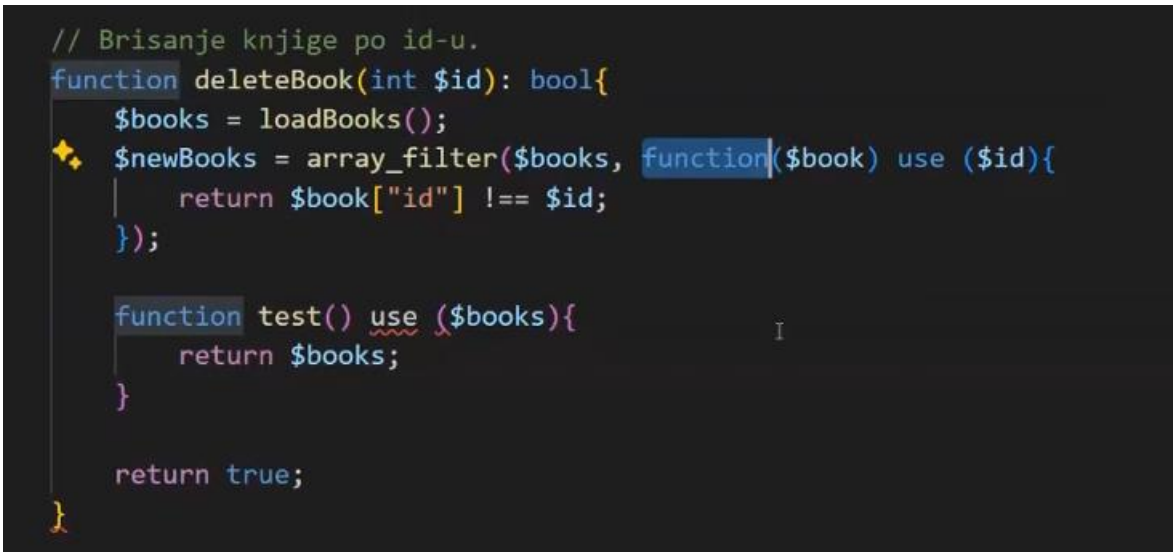
Kod update funkcije može se koristiti referenca na $books, kad mijenjamo u funkciji, mijenjamo i polaznu datoteku **referenca &book**

Nulliš operator ?? upitnika , pitaju je li null u ovoj vrijednosti, ako je null uzmi vrijednost s desne strane $book [„title“], ako nije null uzmi vrijednost parametra $title

$book [„title“]=$title ?? $book [„title“] (to je i vrijednost koja je bila unutra, ako netko nije poslao parametar, ako parametar postoji uzmi vrijednost parametra $title)

```php
foreach($books as $key => $book){
    if($book["id"] === $id){
        $books[$key]["title"] = $title ?? $book["title"];
        $books[$key]["author"] = $author ?? $book["author"];
        $books[$key]["year"] = $year ?? $book["year"];
        saveBooks($books);
        return true;
    }
}
```

Use ne funkcionira kada imamo imenovane funkcije, radi samo kod anonimnih funkcija, vraća sve knjige koje nisu obrisane.



```php
// Brisanje knjige po id-u.
function deleteBook(int $id): bool{
    $books = loadBooks();
    $newBooks = array_filter($books, function($book) use ($id){
        return $book["id"] !== $id;
    });

    function test() use ($books){
        return $books;
    }

    return true;
}
```

2. SUPERGLOBALNE VARIJABLE – interakcija s korisnikom

Varijable koje žive uz cijelu skriptu i dostupne su odasvuda

$GLOBALS

https://www.php.net/manual/en/language.variables.superglobals.php

prazan array ili NULL je false

HTML

Link u **html <a>** mora imati atribut **href** u kojem je putanja, stavlja se parametar u url?querystringparameter, koriste se za slanje jednostavnih podataka

$_SERVER – nema garancije da će web server vratiti sve informacije koje su navedene u dokumentaciji

https://www.php.net/manual/en/reserved.variables.server.php

$_SERVER['PHP_SELF'] – gleda gdje se dogodio request

$_REQUEST – sadrži podatke koji su poslani putem metode $_GET, $_POST, $_COOKIE

https://www.php.net/manual/en/reserved.variables.request.php

$_GET – ako nije prazna, a radi se o asocijativnom arreyu, dobiti ćemo unutra informacije o querystring parmetrima

Sve što dođe kroz url je u principu neki GET zahtjev

GET – opasnost cross side scripting, potrebna je sanitizacija podataka – zamijenitit sve znakove koji su problematičnoi <>, oni znače

• **Oprez**: podaci koje je unio korisnik neće biti sanitizirani

Ako želimo podatak upisati u bazu treba ga sanitizirati – pretvara <>“‘ – **htmlentities** – riješen crosside scripting

Ne dozvoliti direktno pisanje ubazu bez čišćenja

$_POST – omogućuje da se pošalju podaci, kreira se forma i obavezno mora imati metodu post, **action** je prazan ako se radi unutar iste datoteka ili mu se šalje druga datoteka u kojoj se podaci obrađuju

U HTML kod definicije text inputa staviti **require** ako želimo da polje bude obavezno za unos
Kod testiranja na cijelom obrascu možemo ugasiti validaciju kod **definicija forme- novalidate**
e-mail validator
datoteka je binarni podatak i nije poslana u requestu, poslan je samo naziv datoteke jer se šalju samo text podaci u zahtjevu
-da bi se mogle poslati datoteke mora biti uključeno enctype“multipart/form-dana“):

```
<td width="50%" valign="top">
    <h1>Add Book</h1>
    <form action="" method="post" novalidate enctype="multipart/form-data">
        <p>
            <label for="title">Title:</label>
            <input type="text" name="title" id="title" required>
        </p>
        <p>
            <label for="author">Author:</label>
            <input type="text" name="author" id="author" required>
        </p>
        <p>
            <label for="year">Year:</label>
            <input type="number" name="year" id="year" required>
        </p>
        <p>
```

OBAVEZNO atribut „NAME“ MORA BITI
Kad nakon posta odberemo refresh i onda se još jednom pošalje POST, jer je request na istu datoteku, bino je da se rade provjere u PHP
Ako se dogodi Post onda želimo obraditi taj post u
Iza obrasca ubacujemo PHP i pitamo je li request metoda POST
u $_POST varijabli nema podataka o datoteci, podaci se nalaze u superglobalnoj varijabli $_FILE

```
array(3) {
  ["title"]=>
  string(0) ""
  ["author"]=>
  string(0) ""
  ["year"]=>
  string(0) ""
}
array(1) {
  ["cover"]=>
  array(6) {
    ["name"]=>
    string(25) "The Lord of the Rings.jpg"
    ["full_path"]=>
    string(25) "The Lord of the Rings.jpg"
    ["type"]=>
    string(10) "image/jpeg"
    ["tmp_name"]=>
    string(24) "C:\xampp\tmp\php7A41.tmp"
    ["error"]=>
    int(0)
    ["size"]=>
    int(120241)
  }
}
```

```php
<?php
    if ($_SERVER["REQUEST_METHOD"] === "POST") {
        $title = htmlentities($_POST["title"]);
        $author = htmlentities($_POST["author"]);
        $year = htmlentities($_POST["year"]);

        $cover = $_FILES["cover"];

        if ($cover["error"] === UPLOAD_ERR_OK) {
            $targetDir = "uploads/";
            $targetFile = $targetDir . time() . "_" . basename($cover
            ["name"]);

            if(move_uploaded_file($cover["tmp_name"], $targetFile)){
                $id = addBook($title, $author, $year, $targetFile);
```

Minify Ln 70, Col 41 (7 selected) Spaces: 4 UTF

Ako netko ne uploda file imamo grešku i to dobije obrada UPLOAD_ERR_OK

Header – mijenja post metodu u get kod ponovnog refresha više ne radi post nego je zadnji get

```php
if(move_uploaded_file($cover["tmp_name"], $targetFile)){
    $id = addBook($title, $author, $year, $targetFile);
    echo "<p>Book added with id: $id</p>";
    header("Location: ".$_SERVER['PHP_SELF']);
} else {
    echo "<p>Failed to add new book!</p>";
```

MIME type – treba definirati zbog sigurnosti da ne mogu uplodadi npr. dll file
Problem veličine datoteke – na serverskoj strani provjeriti veličinu datoteke, prije nego što pohranimo datoteku na server mi obradimo datoteku GD image library – rad imagima

Sustav autorizacije – server svakom klijentu šalje nekakav session ID, ili sustav kroz token JVT
-problem sessiona je da nam se ukrade session ID (opasnost da nam netko pogodi, izgenerira, snifing prometa (ako nema https), cros side scripting

Kako bi mogli koristiti $_SESSION (Session varijable) – u indeks.php
stavimo **Session start** ()

Kako spriječiti session hijacking – tako da kod svakog novog zahtjeva, pokretanja stranice generiramo novi session ID

```php
<?php

session_start();
session_regenerate_id();

if ($_GET["login"] ?? false) {
    $_SESSION["username"] = "admin";
    header("Location: index.php");
    exit;
}

echo "Nisi prijavljen!";
```

Kad startamo session starti – dobijemo pristup globalnoj varijabil $_SESSION
Obavezno koristiti Https, vremensko ograničenje trajanje sesije, session se destroya zatvaranjem borwsera

S_COOKIE
Ne možemo pristupiti kolačići java scriptom, nego samo http zahtjev
Setcookie () – ima parametara, destroja se ako vrijeme do kojeg vrijedi je u prošlosti
Mala datoteka koja se spremi na računalo korisnika , veličine do 65kb
Mogu se koristiti da se korisnik na svojem računalu ponovno i ponovno logira na računali, treba ih izbjegavati za čuvanje osjetljivih stvari

- Samo na razini određenog dijela aplikacije (samo za administratore, samo za javni dio ...)
- Može na razini domene (poddomena na kojoj mogu djelovati)
- Secure – mora biti uključen neki certifikat, cijela aplikacija mora funkcionirati na https:// protokolu -kolačići su kriptirani
- Zadnje flag je httponly flag - sprečav da se sa javascriptom dohvati taj kolačić – onemogućeno je da javascript čita cookie

```php
session_start();
session_regenerate_id();

if ($_GET["login"] ?? false) {
    $_SESSION["username"] = "admin";
    setcookie("username", "admin", time() + 3600, "/", "", false, true);
    header("Location: index.php");
    exit;
}

echo "Nisi prijavljen!";
```

Može se čitati kolačić putem superglobalne varijable **$_cookie**, PHP može čitati

```php
var_dump($_COOKIE["username"]);
```

Ako radimo u private modu – cookie se ne spremaju

3. OOP

Klasa - je nacrt iz kojeg možemo stvoriti neki objekt
Možemo sve raditi sa objektima i sve može biti objekt

Apstrakcija – uzmete objekt (korisnika) koji predstavlja neki objekt
Korisnik unutar jedne aplikacije ima jedan set karakteristika, a unutar druge aplikacije drugi set karakteristika koje su bitne karakteristike unutar aplikacije koju radimo

Enkapsulacija – cilj je da i podaci i procedure koje upravljaju tim podacima su unutar objekta, želimo učahuriti te podatke unutar objekta, izvana pristup nije moguć, kontrolira se pristup tim podacima kroz definirana sučelja (metode koje objekt mora imati kako bi mogao komunicirati s drugim objektima)

Nasljeđivanje – jedna klasa nasljeđuje atribute i metode druge klase

Polimorfizam – Omogućuje da se isto ime funkcije koristi za različite tipove radnji, npr radi naplatu ali je koncept naplata te drugačiji

Interface – mora postojati sučelje, ako se ispravno pozove metoda preko sučelja može komunicirati s klasom

Osnove OOP

- Za imena klasa trebali bi koristiti pascal case **MyCar**
- Može se stvoriti novi objekt sa clone, mijenja se novi objekt
- Može se kopirati objekt u novi objekt (clone)

- Osim instancirnja (**new**) se može klonirati i kopirati objekt**(=),** kopiranje objekta je kopija objekta, a kloniranjem se stvara sasvim novi objekt (**clone**)

Konstruktor – magične metode

Ugrađene su kroz PHP, ne moraju se koristiti, konstruktor omogućava pametnu inicijalizaciju objekta, **new** aktivira funkciju konstruktor, zadaća je da izgradi objekt

Ima parametre kao i funkcija, parametri se pune podacima kada pozovemo klasu kod instanciranja, šalju se podaci i oni se odmah primjene unutar objekta

Destruktor

je funkcija koja se koristi kad se objekt briše, makne, ne treba jer postoji garbage collector

Kontrola pristupa

- Kako bi mogli primijeniti učahurivanje,

Access modifieri

- **Public** - Kad je neka metoda public, može joj se pristupiti u samoj klasi, ali i izvan klase
- **Protected** – svojstvo i metoda dostupni su samo unutar klase i klase koji ju nasljeđuju
- **Private** -svojstvo i metoda je dostupna samo unutar klase

$this – pristupi ovoj klasi i u toj klasi pristupi svojstvu „Brand“

__get() i **__set()** su metode kojima kontroliram pristup svojim svojstvima – getter i setter metode

Magic methods - izbjegavati magic metode – Laravel ih koristi

Geteri i Seteri moraju biti javno dostupni (public)

__tostring – omogućuje ispisivanje stringa iz objekta – magična metoda

Nasljeđivanje – modularnost – kod konstruktora poziva se konstruktor svojih roditelja

```php
class Student extends Osoba {
    private string $jmbag;

    public function __construct(string $ime, string $prezime, string $jmbag){
        parent::__construct($ime, $prezime);
        $this->jmbag = $jmbag;
    }
}
```

Student može imati svoju metodu _toString i to je polimorfizam – dva ista imena, ali sa dvije različite stvari koje treba raditi

```php
    }

    public function __toString(){
        return parent::__toString() . "-" . $this->jmbag;
    }
}
```

Sučelje je obaveza za klasu da implementira neku metodu

- Omogućuju da znamo što naša klasa radi (**interface**)
- Ne moramo strogo definirati koji objekt ćemo primiti
- Samo ime metode i scope metode, trebaju se koristiti samo za javno dostupne metode,
- Sučelje ne definira kako će se metoda ponašati (potpis što treba klasa implementirati)
- Sučelja se mogu nasljeđivati (ali nije preporučljivo koristiti) – krši se pravilo interface segregation
- Omogućuje da možemo komunicirati prema van s drugim objektima

```php
function printArea(Shape $shape) {
    echo 'Površina: ' . $shape->getArea() . PHP_EOL;
}

$circle = new Circle(5);
$rectangle = new Rectangle(3, 4);
```

Sve klase koje imaju implementirano određeno sučelje mogu se poslati kao parametar u funkciji

Bazna klasa – služi kao temelj da ostale klase mogu naslijediti tu klasu, ali se ne može instancirati u objekt, ta klasa ne bi trebala biti kao obična klasa, apstraktna klasa je bazna klasa koja bi se trebala koristiti kao temelj, koristi se kao osnova

Kreirati skicu za ostale klase i prisiliti ostale klase da implementiraju određene metode, prisiljavanje se može izbjeći, može biti obvezujuća

Razlika između sučelja - u sučelju su samo definirane metode koja klasa treba kreirati, a kad se nasljeđuje apstraktna sve one metode koji imaju u definiciji **abstract** moraju biti implementirane u klasi koja ju nasljeđuje, dok ostale metode klasa koja nasljeđuje apstraktnu klasu preuzima od same temeljne klase. Dobra praksa je da apstraktna klasa implementira obvezujuće metode za klase koje ju nasljeđuju, ne bi trebalo miješati funkcionalnosti koje neka klasa treba naslijediti s obvezujućim metodama (za to koristiti interface)

Class Dog extend Animals (ako je Animals apstraktna klasa)

Class Dog extend Animals implements (neki interface) – kad se dodaje i klasa i interface

Apstraktna klasa može naslijediti drugu klasu, ali trebalo bi izbjeći to jer je apstraktna klasa temelj

Statičke metode (funkcije) – mogu se pozivati na razini klase bez da se klasa instancira u objekt

Ako želimo zabraniti mogućnost istanciranja klase u kojoj su navedene static metode tu klasu trebamo proglasiti Abstract (više nije po default onemogućen pristup metodi koja je **static** ako instanciramo takvu klasu.

Definirati metode na razini klase i mogu se pozivati na razini klase, ne moramo stvarati objekt, dostupna su na razini klase.

Koriste se:

Helper klase – ne moramo instancirati objekt da bi došli do funkcionalnosti, nego se referenciramo direktno na statičku metodu

Za definiranje paterna potrebno je definirati određene paterne -**Singleton**

Kad moramo raditi s nečim što nam treba, bez da instanciramo klasu u objekt

```php
class Utils{
    public static function generateRandomString(int $length): string{
        $characters = '0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ';
        $charactersLength = strlen($characters);
        $randomString = '';
        for($i = 0; $i < $length; $i++){
            $randomString .= $characters[rand(0, $charactersLength - 1)];
        }
        return $randomString;
    }
}

echo Utils::generateRandomString(10) . PHP_EOL;
```

UUID ili GUID koriste se za generiranje jedinstvenih identifikator (PHP uniqid).

Enumeratori:

-vrsta klasa koja ima definirani set vrijednosti koje su dozvoljene, kada želimo ograničiti unos podatka na točno određene vrijednosti (npr. unos spola)

```php
enum Gender{
    case MA;
    case FE;
    case OT;
}

class User{
    private string $name;
    private Gender $gender;

    public function __construct(string $name, Gender $gender){
        $this->name = $name;
        $this->gender = $gender;
    }
}

$user = new User("Mark", Gender::MA);
```

Magic method __get je umjesto klasičnog gettera, radi za bilo koji get, ali sad nemamo kontrolu, radi na svakom privatnom proprety

```php
    public function __get($name){
        return $this->$name;
    }
```

Backed Enumeratori koji nam omogućavaju da definiramo vrijednost, rade jedino i isključivo sa integerom i stringom, enumerator vrati vrijednost koja odgovara određenom slučaju. Basic enumeratori – ne možemo nigdje ispisati zato se koriste Backed enumeratori.

```php
enum Gender: string{
    case MA = "Male";
    case FE = "Female";
    case OT = "Other";
}
```

Primjer sortiranja: **Usort** radi na principu quick sorta

Arrow function: => Umjesto **return**

```php
usort($events, fn($a, $b) => $a->sortBy("startDate") <=> $b->sortBy("startDate"));
```

```php
usort($events, fn($a, $b) => $a <=> $b);
```

4. AUTOLOADER

Imenski prostori – organizacija datoteka (namespaces) – što bolja organizacija koda

- Problem klasa koje se isto zovu, i datoteke koje se isto zovu

Kreirati foldere –

Mapa src – mapa gdje držimo sav naš kod (kalse, enumeratore ...)

PSR-4 standard koji se koristi za autoload – ime datoteke se zove isto kao i ime resourcea koji je unutra pohranjen

Ako npr. trebamo negdje enumerator Event Type, navedemo namespce, a autoloader se automatski uključi

Autoloader ulazi u src folder i definira imenski prostor, koji je vezan uz strukturu datoteke, isčupa datoteke i uključi ih tamo gdje treba

Nema problema s includanjem, hoćemo koristiti imenske prostore i da nam se automatski uključi

App je autoloadre za src (PSR-4)

Ime vendora + prefix + arhitektura, u istom imenskom prostoru dvije klase se ne mogu isto zvati

Naš prefix je App i on je vezan uz mapu **src**, ostatak imenskog prostora mora se podudarti sa strukturom

```php
namespace App\Classes;

class Organizer extends Person{
    static private int $nextId = 1;
    private array $events = [];
```

Naša klasa Organizer zivi u imenskom prostoru App\Classes

use App\Interfaces\Confirmable;

Autoloader se mora pobrinuti da pomoću use pronađe datoteku na File sistemu i napravi include

Bitno je imati vezu između imenskih prostora i strukture**. Ako su u istom imenskom prostoru ne treba pisati use, autoader to sam prepoznaje.**

Src treba kreirati zato jer PHP ima svoj imenski prostor **– Globalni imenski prostor**

Ako želimo koristiti nešto iz PHP moramo koristiti to iz njegovog (PHP) imenskog prostora,

Primjer **(DateTime – klasa, moramo pisti Use DateTime);**

Kad gledamo strukturu naše klase uvijek na početku je definiran imenski prostor, zatim direktive Use i nakon toga ide naša klasa.

Još jednom ime datoteke i klase ili bilo kojeg našeg resoursa mora biti jednako, jer na temelju toga radi autoloder. Da nema te logike onda bi u autoloaderu morali sve posebno pisati.

Još moramo sve uključiti u Indeks.php (indeks je u globalnom PHP prostoru tko da mu npr. ne moramo uključivati imenske prostore PHP-a (npr. DateTime)

PHP ima funkciju koja automatski prilikom prvog učitavanja odmah sve poveže, odradi registraciju svih tih putanja

Spl_autoload_register – predam joj callback funkciju koju će koristiti i pokrenuti tako da kad naleti na use ispravno pretvori u putanju te datoteke i ispravno uključi

Autoloader – Callback Clases, autoloader dobije informaciju preko imenskog prostora jedan dio putanje, zaprima iz imenskog prostora,

Ne želimo da autoloader pukne jer nije našao datoteku

Pokrenemo Indeks.php gdje je include Autoloadera, dohvatio je iz use imenski prostor i ime nekog resourca i na temelju te informacije želimo složiti putanju koja će doći do mape classess i složiti putanje

$prefix="App\\" – moramo zamijeniti s našim src (\\ stavlja se dvostruki jer ako stavimo jednostruki misli da želimo ignorirati zadnji znak)

```
$prefix="App\\";
$base_dir=__DIR__ . "/../src/";
```

Definiran je base_dir i uspostavljena veza između autoloada i src i autoload zna kamo treba ići

DIR - gleda gdje se nalazi vendor (moramo izaći iz vendora).

```
$len=strlen($prefix);
if (strncmp($prefix, $class,$len)!==0){
    return;
}
```

Strncmp – binarno uspoređuje dva stringa u duljini len, ako su jednaka vraća 0, ako su različita izlazimo iz autoloada

5. KONTROLNE STRUKTURE

- da bi dobili objekt iznimke mora postojati klasa iz koje se stvori taj objekt, omogućuje kontrolirano izvršavanje programa bez da se zaustavi daljnje izvođenje programa
- ključne riječi **try** – blok koda u kojem se pokušava izvesti neki kod tj. Dok se izvodi testira se za greške ako se unutar bloka dogodi neka greška može se reagirati, da bi se dogodila neka greška koristi se ključna riječ **Trow** čija je zadaća da baci iznimku. Ne moramo to mi raditi, to može doći iz sam sintakse PHP
  catch – iznimka se hvata u tom bloku i obradi se
  finaly - je opcionalan i koristi se neovisno o tome što se dogodilo, uvijek se izvrši taj kod

  Exception je bazna klasa za sve iznimke, implementira sučelje Throwable
  ErrorException – koristi se za obradu PHP grešaka kao iznimki, razdvojena priča, extenda exception
  Bilo koja iznimka ako se dogodi biti će uhvaćena – nešto što PHP generira kada dođe do php grešaka (warning, require),

  Specijalizirane iznimke – **InvaliAargumentException** – svi se na kraju vrate na baznu klasu, ako ne želimo uhvatiti specifične iznimke možemo uvijek uhvatiti baznu iznimku iz Exception

- Možemo imati i prilagođene iznimke Logika prilagođenih iznimki je to da znamo kakva se iznimka dogodila
  Kad se dogodi greška u **Trow** funkcija se zaustavlja,
  Kad se funkcija pozove u **try** bloku sav njezin kod se validira na greške, ako ne postoji greška sve ok, ako postoji greška **catch** kod se aktivira i hvata grešku i ispisuje obavijest o greški
  Exeption driven development – potiče programere da misle o greškama unaprijed, prije nego počnu pisati kod
  Pišu se testovi koji provjere je li promjena koda utjecala na greške

```php
function processFile($filename) {
    try {
        $file = fopen($filename, "r");
        if (!$file) {
            throw new Exception("Datoteka ne može biti otvorena.");
        }
        // Obradi podatke iz datoteke
        while (!feof($file)) {
            $line = fgets($file);
            // Pretpostavimo da postoji neka specifična logika obrade ovdje
        }
        fclose($file);
    } catch (FileNotFoundException $e) {
        // Specifična obrada za slučaj kada datoteka nije pronađena
        echo "Greška: " . $e->getMessage();
    } catch (Exception $e) {
        // Opća obrada za sve ostale iznimke
        echo "Došlo je do greške: " . $e->getMessage();
    } finally {
        echo "Proces obrade je završen.";
    }
}
```

Moguće je imati više catch blokova gdje imamo propadanje

Bazni exception mora biti zadnji u nizu

U imenik klase za Exception na kraju uvijek staviti Exception

- Kad netko želi staviti depozit na račun i ispitati ako je taj depozit negativan

Prednost iznimaka – znamo gdje se potencijalno dogodila iznimka, možemo doći do informacije u kojoj datoteci se dogodila iznimka i u kojoj liniji koda se dogodila greška

Ako nemamo try blok dogodi se fatal error – koji govori imaš neuhvaćenu iznimku

Ako se ne dogodi greška izvrši se try blok

Svaki dio koda koji nešto radi, ali moramo obraditi svaku iznimku koju taj kod radi

Kad koristimo nešto pogledamo da li može ta funkcija generirati grešku

```php
class AccountException extends Exception{}
Class Account{

    private float  $balance =0;

    public function __construct(bool $isSavings=false){
        if($isSavings){
            throw new Exception ("savings account not supported");
        }
    }

    public function deposit (float $ammount):void {
        if ($ammount  <=0) {
            throw new AccountException("Amount must be greter than 0");
        }
        $this->balance+=$ammount;
    }
}
$account =new Account();
try{
    $account->deposit(-100);
    var_dump ($account);
}catch (AccountException $e)
{
echo $e->getMessage();
}
```

Ako konstruktor može generirati grešku, a ne stavimo instanciranje objekta u try catch blok onda dobijemo fatal error **Uncaught Exception**

```php
<?php
class AccountException extends Exception{}
Class Account{

    private float  $balance =0;

    public function __construct(bool $isSavings=false){
        if($isSavings){
            throw new Exception ("savings account not supported");
        }
    }

    public function deposit (float $ammount):void {
        if ($ammount  <=0) {
            throw new AccountException("Amount must be greter than 0");
        }
        $this->balance+=$ammount;
    }
}

try{
    $account =new Account(true);
    $account->deposit(-100);
    var_dump ($account);
}catch (AccountException $e)
{
echo $e->getMessage();
}catch(Exception $e){
    echo "General exception: " . $e->getMessage();
}
```

Svaki kod koji može generirati iznimku mora biti u try bloku i mora iznimka biti obrađena .

Ako imamo više catch blokova moramo paziti na redoslijed i uvijek se ide od childe, zadnji mora biti exception. Koja iznimka se prva dogodi try blok prestaje s radom. Npr, ako se dogodi iznimka kod instanciranja klase ova druga iznimka s iznosom depozita neće ni biti ispitana jer try blok prestaje s radom

Može se proslijediti na način da se nakon obrade u catch bloku doda **throw $e** i da se greška zapisuje u log.

6. DIZAJN OBRASCI I SOLID PRINCIPI

- Svi paterni koji postoje, a ne samo za PHP (https://refactoring.guru/design-patterns)

- Standardizirana rješenja za česte probleme koje susrećemo kod dizajna softvera, neka vrsta template po kojem napišemo naš kod i on je spreman za izvođenje

- Problem kompatibilnosti sa različitim stranama (npr. payemetn getway – google payment, paypall – svatko radi to na svoj način)

  Addapter patern – obrazac po kojem programiramo

Vrste dizajn obrazaca

Craetional – Singleton, Factory – obrazac koji omogućuje da stvorimo samo jedan objekt (klasa je definirana po singletone obrascu)

Structural – olakšavaju dizajn arhitekture sustava (Adapter, Facade – fasada objekta tako da netko na vidi strukturu objekta, a fasade omogućuje da sve funkcionalnosti objekta rade

Behavioral – usmjereni su na komunikaciju između objekata Observer, Strategy

Singleton obrazac – osigurava da klasa ima samo jednu instancu i pruža globalnu točku pristupa toj instanci – kod baza konstruktor je privatan i stvar se static – **Connect na bazu**

7. SOLID PRINCIPI

S: klasa bi trebala imati samo jednu zadaću ili odgovornost (Single Responsibility)

O: software bi trebao biti otvoren za proširenja, ali zatvoren za izmjenu

L: objekti u programu bi trebali biti zamjenjivi s instancama njihovih podtipova

I: ni jedan klijent (klasa) ne bi trebala ovisiti o metodi koju ne koristi

D:visokorazinski moduli ne bi , kako se dva objekta ponašaju kada jedan injectamo u drugi

Dependency injection je gdje imamo dva objekta gdje jedan objakt injectamo u drugi da bi ga ovaj drugi mogao koristiti (najčešće kroz metodu ili kroz konstruktor)

Single responsibility:

```php
// Loš pristup: Klasa "User" upravlja korisničkim podacima i logikom slanja e-pošte
class User {
    public $email;
    public $username;

    public function sendEmail($content) {
        // Logika za slanje e-pošte
    }
}

// Bolji pristup: Razdvajanje odgovornosti
class User {
    public $email;
    public $username;
}

class EmailService {
    public function sendEmail($user, $content) {
        // Logika za slanje e-pošte
    }
}
```

Razdvajanje odgovornosti – upravljanje korisnicima, slanje maila

Open / Closed Principle

Open/Closed Principle

Princip otvorenosti/zatvorenosti kaže da bi software entiteti (klase, moduli, funkcije itd.) trebali biti otvoreni za proširenje, ali zatvoreni za izmjene.
To znači da bi trebali moći dodati nove funkcionalnosti bez mijenjanja postojećeg koda.

PaymentProcessor klasa je zatvorena za izmjene ali otvorena za proširenje kroz implementaciju ***PaymentGateway*** interfacea.

```php
interface PaymentGateway {
    public function processPayment($amount);
}

class PaypalPayment implements PaymentGateway {
    public function processPayment($amount) {
        // Proces plaćanja preko PayPal-a
    }
}

class CreditCardPayment implements PaymentGateway {
    public function processPayment($amount) {
        // Proces plaćanja kreditnom karticom
    }
}

class PaymentProcessor {
    private $paymentGateway;

    public function __construct(PaymentGateway $paymentGateway) {
        $this->paymentGateway = $paymentGateway;
    }

    public function pay($amount) {
        $this->paymentGateway->processPayment($amount);
    }
}
```

PeymentProcessor je rađen na open/close principu **– u konstruktor šaljemo intreface,** a ne klasu CreditCardPayment ili neku drugu, sada ako u interface dodamo još neku metodu, ne ovisi o metodu za plaćanje nemamo problema jer smo vezani na sučelje, a naziv metode plaćanja šaljemo kroz konstruktor, konstruktor kaže da u parametru mora dobiti objekt koji ima implementirano PayementGateway sučelje

Liskov Substitution Principle

Liskovljev princip supstitucije kaže da bi objekti programa trebali biti zamjenjivi s objektima njihovih podtipova bez utjecaja na točnost programa.
To znači da bi derivirani tipovi trebali moći zamijeniti svoje bazne tipove.

U ovom primjeru, korištenje ***Ostrich*** klase može izazvati probleme jer nojevi ne mogu letjeti. Ovaj dizajn krši LSP.

```php
class Bird {
    public function fly(){
        echo "Leti";
    }
}

class Duck extends Bird {}

class Ostrich extends Bird {
    public function fly() {
        throw new Exception("Ne može letjeti");
    }
}
```

Ne može se zamijeniti bazna klasa sa child klasom, patka leti, ali noj ne leti – ako je točno da child može u bilo kojem trenutku zamijeniti parenta držimo se tog principa

Interface Segregation Principle

Princip segregacije sučelja sugerira da klijenti ne bi trebali biti prisiljeni ovisiti o sučeljima koja ne koriste.
To znači da bi sučelja trebala biti specifična, a ne generička.

Ovaj dizajn krši ISP jer ***RobotWorker*** ne treba *eat* metodu. Bolje bi bilo razdvojiti ova sučelja.

```php
interface Workable {
    public function work();
    public function eat();
}

class HumanWorker implements Workable {
    public function work() {
        // radi
    }
    public function eat() {
        // jede
    }
}

class RobotWorker implements Workable {
    public function work() {
        // radi
    }
    public function eat() {
        // Ova funkcija ne treba robotu
    }
}
```

Ne smijemo tjerati klasu da implementira metodu koju ne koristi. Razdvojiti sučelja !!!

Dependency Inversion Principle

Princip inverzije ovisnosti navodi da bi visoko razinski moduli ne trebali ovisiti o nisko razinski modulima. Oba bi trebala ovisiti o apstrakcijama.

FileManager ne ovisi direktno o ***DatabaseStorage***, već o ***Storage*** sučelju, što omogućava fleksibilnost i manju povezanost komponenti.

```php
interface Storage {
    public function save($data);
}

class DatabaseStorage implements Storage {
    public function save($data) {
        // Spremi podatke u bazu
    }
}

class FileManager {
    private $storage;

    public function __construct(Storage $storage) {
        $this->storage = $storage;
    }

    public function saveFile($data) {
        $this->storage->save($data);
    }
}
```

Ako imamo modul koji upravljaju datotekama i storage , file manager ne bi smio ovisiti npr, o database storeagu, ako želim promijeniti spremanje podataka ne smije ovisiti o storeagu,
I storeage i File manager moraju implementirati isto sučelje
File manager u konstruktoru u parmetru ne prima Database storage nego tip intreface Storage, a to znači da mu u parametar možemo poslati bilo koji storage koji implementira sučelje storage.
Na storeagu sigurno živi metoda Save jer ona živi u interface Storage

8. KONEKCIJE NA BAZU

mysql_connect – je zastario

mysqli_connect – je samo za sql bazu

PDO -class (PHP database object) – reprezentira konekciju na server – pitanje da se napiše jedna konekcija sa PDO i jedna s **MYSQLI**

Pdo ima KONSTRUKTOR – koji prima tri parametra
New PDO

- Dns - **"mysql:host=localhost;dbname=adventureworkshop;charset=utf8"** (Databse driver
- Username,
- Password

PDO klasa ima metode **prepare, query**

```php
<?php

$pdo = new PDO("mysql:host=localhost;dbname=adventureworkshop;
charset=utf8", "root", "awdawr");

$stm = $pdo->query("SELECT * FROM proizvod");
$res = $stm->fetchAll(PDO::FETCH_ASSOC);
var_dump($res);
```

Klasa PDO statement se dobije nakon što se izvrši Query (prepare)

Nedostatak query na pdo je **sql injection**, query nema provjere gdje binda vrijednost s nekim parametrom iz upita - **OPREZ**

Bolje je **prepare**, onda ne injectamo direktno parametar, nego u prepare statement koristimo placeholdere, koji mogu biti pozicijski i bitno je da ispravno bindamo te vrijednosti po pozicijam ili postoj imenovani, uglavnom se koriste pozicijski.

Nakon pripreme radimo execute. Kod execute se šalje bind svih parametara (array) i uglavnom su pozicijski.

Ne smije se direktno u prepare gurnuti vrijednost koju pošalje korisnik, sama prepare ne radi sprečavanje, execute kod spajanja prepara sa placeholderom sprečava sql injection ,rezultat je asocijativni array

```php
$pdo = new PDO("mysql:host=localhost;dbname=adventureworkshop;
charset=utf8", "root", "");
$id = "1 OR 1=1";
$stm = $pdo->prepare("SELECT * FROM proizvod where IDProizvod = ?");
$stm->execute([$id]);
$res = $stm->fetchAll(PDO::FETCH_OBJ);
var_dump($res);
```

Ako ne želimo kod dohvata uvije pisati PDO:FETCH_OBJ, to navodimo u konekciji, pod options koji prima konstruktor konekcije

```php
$pdo = new PDO("mysql:host=localhost;dbname=adventureworkshop;charset=utf8", "root", "", [
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_OBJ,
]);

$id = "1 OR 1=1";
$stm = $pdo->prepare("SELECT * FROM proizvod where IDProizvod = ?");
$stm->execute([$id]);
$res = $stm->fetch();

var_dump($res);
```

KORISTITI UVIJEK PREPARE STATEMENT I ONDA EXECUTE, Fetch() - vrati uvijek samo prvi zapis, ako dohvatimo više podataka iz baze uvijek dobijemo samo jedan zapis DOHVATI PRVI ZAPIS, A FetchALL() – uvijek gradi array DOHVATI SVE ZAPISE, preporuka je koristiti FatchALL

ORM – ELOQUENT (Laravel) – da ne moramo svaki puta pisati sve ove prepare, execute, fetch

9. COMPOSER

- Composer self-update
- Package manager kojim možemo upravljati paketima
- Kad radimo s php kodom ako radimo vlastiti neko kod dužni smo ga održavati, kako ide nova verzija php kod mora biti kompatibilan s novim verzijama, da li postoji neki sigurnosni propust it. Kada radim s paketima kad koristimo tuđi kod i to se zapakira kao paket i možemo ga koristiti
- Kako upgradati taj kod, vlasnik je dužan izdavati zakrpe, provjerava se vjerodostojnost paketa https://packagist.org/

composer init – omogućava da koristimo njegov autioloader – PSR-4 standard je u principu autoloader https://www.php-fig.org/psr/psr-4/

Bitan je namespace – App (to naprave svi), Prvo ime vendora, a onda ime projekta (\simfoni\crm)

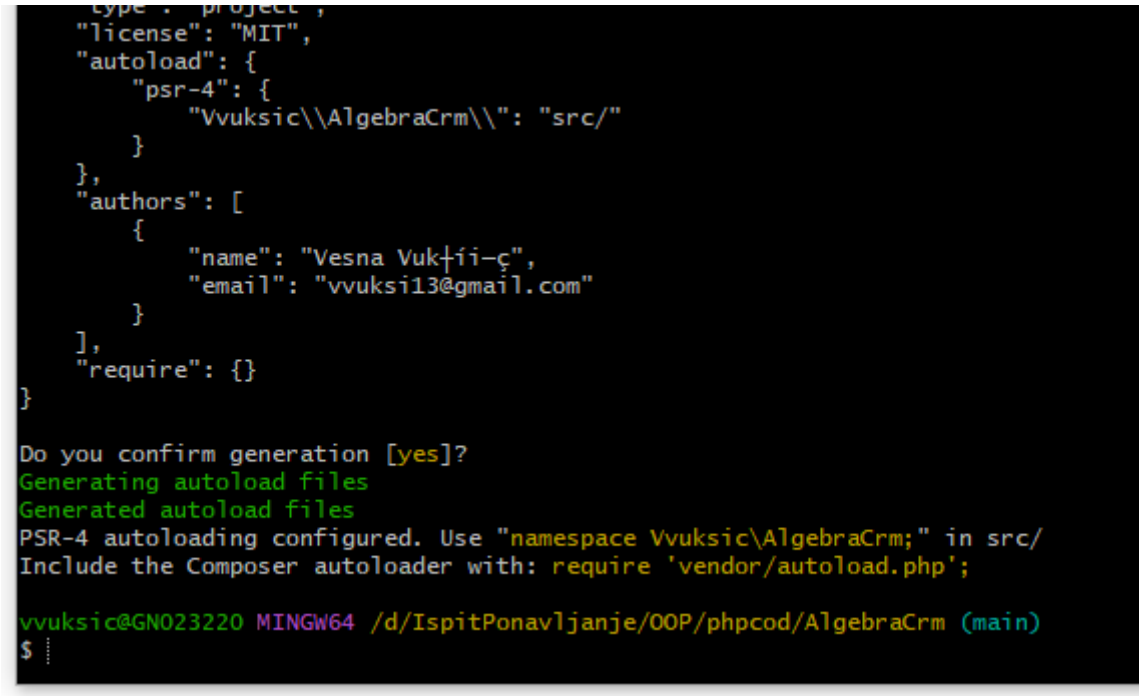
INSTALACIJA:

Mi razvijamo vlastiti – ostavimo defult

Licenca MIT – popustljiva licenca

Da li želim definirati svoje ovisnosti - ne

PSR-4 autoload – mapira na mapu **src i uključuje autolader vendor/autoload.php**

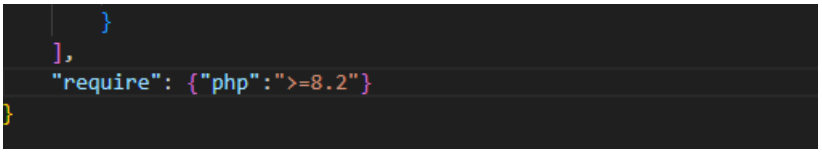


Kreirao je **composer.json** datoteku, ako iz kojeg razloga izbrišemo vendor mapu iz composer.json datoteke možemo ju rekonstruirati. Ne smijemo izgubiti datoteku composer.json

Composer.lock – ubrzava proces instalacije ili upgrade nekih paketa ali je nebitan, dobar je jer će brzinski preko njega isčekirati imam to već u cacheu pa ćemo to izvući iz locka, koristi se i kod upgradea, ne dijeli se na git jer možda netko ne naprvi update (composer update)

Mapa **vendor** ne smije ići na **git**

U composer,json u polje require navodimo koju verziju PHP trebamo koristiti, composer kod instalacije paketa provjerava koja verzija PHP-a se zahtijeva



Nakon toga treba podesiti autoloder + umjesto „vvuksic\\AlgebraCrm\\ - staviti „**App“**

- Inicijalizacij GIt-a – **git init**
- **U projektu napraviti mapu .gitignore i u nju staviti sve ono što ne ide na git (/vendor composer.lock)**

Slijedeći korak je uspostava arhitekture – MVC – kako postaviti projekt

10. MVC

M – model

V – pogled

C - controler

Razdvajamo funkcionalnsiti da znamo što tko radi

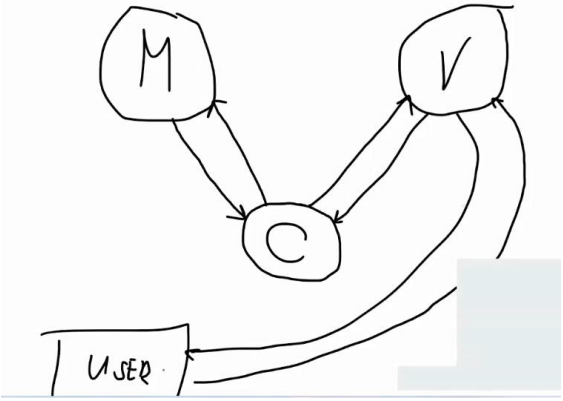
Controler -upravlja zahtjevima

Korisnik pošalje zahtjev koji je pristigao na kontroler, kontroler pita model koji je zadužen za kominikaciju s bazom, u modelu se događa poslovna logika

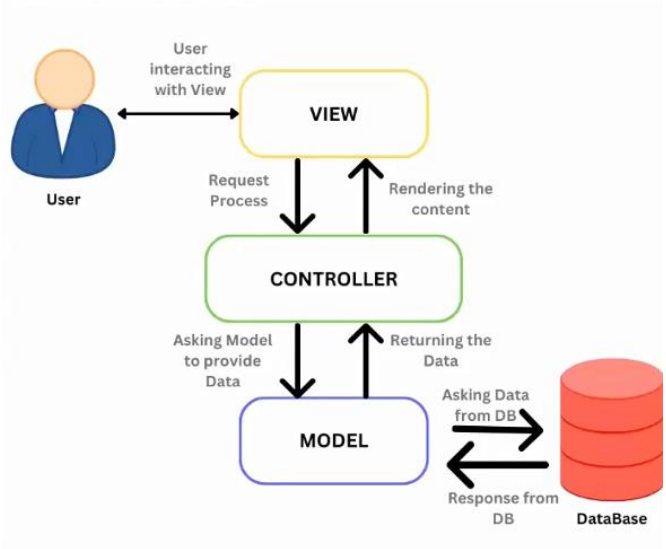
Kontroler samo kaže korisnik je kliknuo na url, trebamo mu nešto vratiti, što trebamo vratitit, npr. nakakve klijente, controler to ne zna i pita model, model kaže da dohvati to iz baze, model može direktno iz modela u ići u pogled Viev (to je rijeđe), drugi način je da model modelirane podatke vraća kontroleru, a kontroler te podatke modelirane nalouda na pogled i te modelirane podatke vrati korisniku

Korisnik do kontrolera dođe preko Pogleda tu mu je ulaz.

.- s pogleda se poslao zahtjev koji je završio na kontroleru

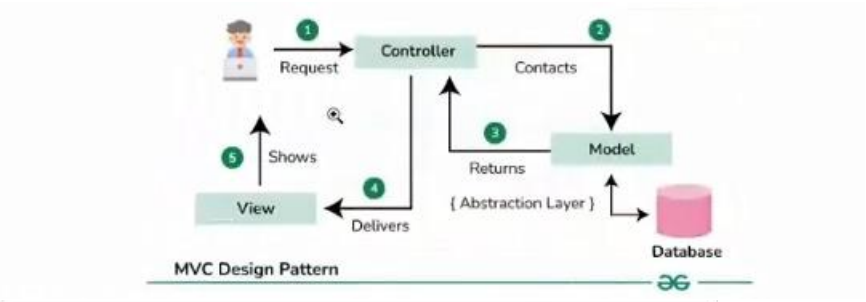


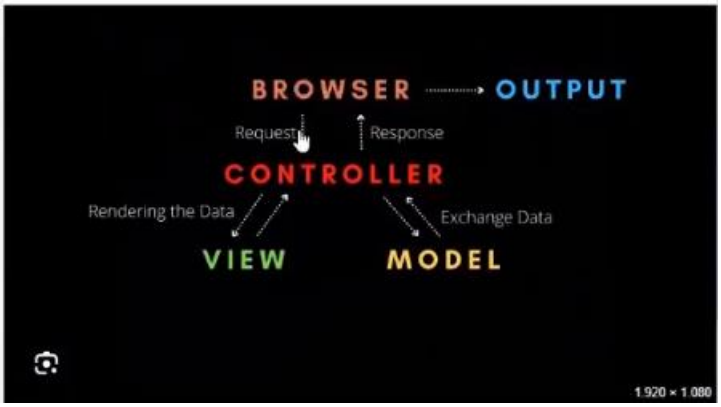
(USER)Korisnik klikne nešto na Pogledu (V) – Pogled šalje zahtjev controleru (C) – controler šalje upit modelu (M) – model pripremi podatke i vrati modelirane podatke controleru (C) – controler ih prikaže na pogledu (V) i vrati pogled korisniku(USER)



- Sa pogleda se šalje request, ali u biti request prvo mora stići na kontroler, moramo imati jednu ulaznu točku

Ovo je ispravnije





Browser radi request na kontroler koji onda pinga modele (exchange Data (MODEL) i Rendering dana (VIEW), priprema podatke i vraća browseru odgovor, model i pogled je spojio u pogled i to je vratio browseru da to browser pokaže

Kako iz browsera poslati zahtjev na controler – controler nije jedan ima ih više, ako imamo jedan kontroler kršimo single responsibility (SOLID) – TREBA NAM Routing – sustav koji će brinuti da na temelju url zna kojem kontroleru poslati zahtjev

- Na serveru se kaže da svi zahtjevi koji dođu na server budu preusmjereni na jednu datoteku – u toj datoteci ćemo uključiti routing i svi zahtjevi će pristizati routeru i on će ih
- prosljeđivati kontrolerima

Kako to izvesti:

- kreirati novi folder **public –** prema van otvorimo smo ovu mapu – otvaram samo datoteku indeks.php, kad upalimo autoloader index će uključiti routing

značiindeks ima zadaću uključi autoloadrer i uključi routing, sve zahtjeve treba redirektati na indeks.php – to se definira na appache (**browser napravi upit na server, server ga usmjeri na index.php, u indexu upalimo router gdje kažemo pročitaj putanju sa koje je došao zahtjev i tu putanju preusmjeri na tu i tu rutu, na toj ruti kažemo zadužen ti je taj kontroler i uzmemo podatke iz zahtjeva i pošaljemo ih kontroleru i onda kontroler dalje radi svoje.**

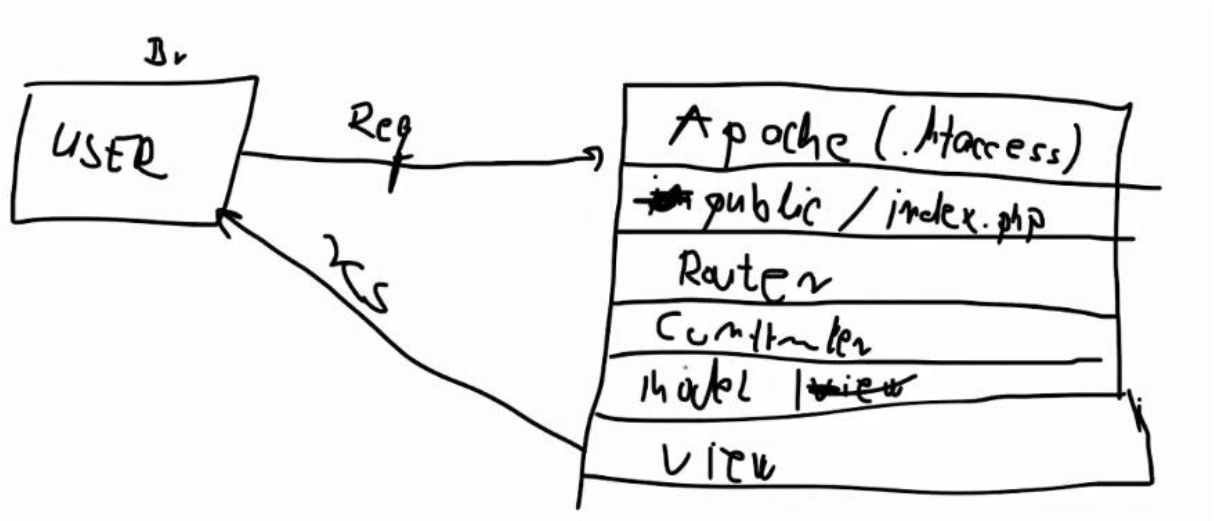
```php
<?php

require __DIR__ . '/../vendor/autoload.php';
```

__DIR__ je parent adresa od one u kojoj je datoteka index.php

.htaccess datoteka na Appache serveru – omogućava da in fly mijenjamo konfiguraciju Appache servera, treba upaliti rewirite mode i upaliti rewrite rule

Želimo kad dođemo na root direktorije želimo da ga appache preusmjeri na public i index.php

U rootu projekta kreiramo .htaccess i sada radi preusmjeravanje na public mapu i indeks.php



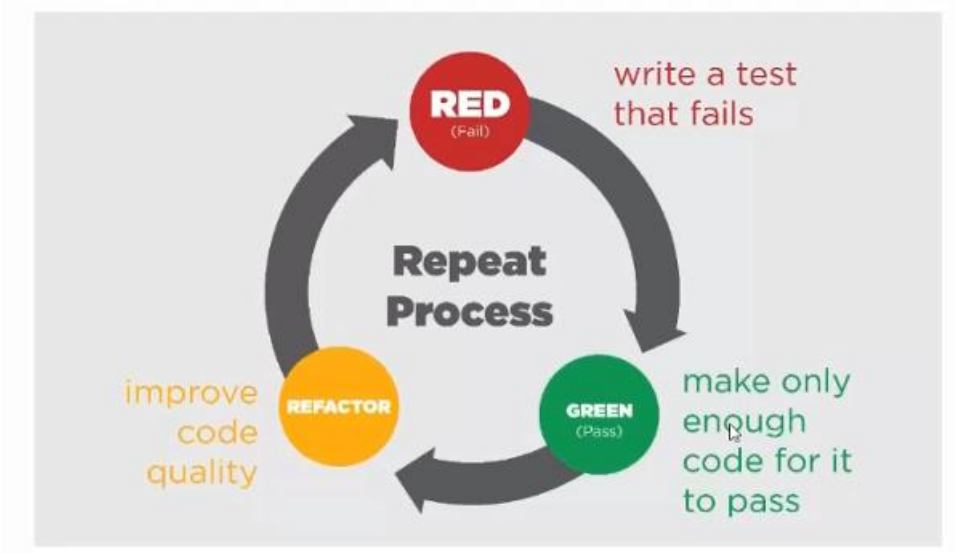
Sad možemo u indeks.php pročitati s kojeg je urla došao zahtjev i preusmjeriti ga pomoću routera na kontroler koji je zadužen za taj url

Kreiramo mapu config i u njoj app.php u koju definiramo manje osjetljive podatke

11. TESTIRANJE

Dva pristupa samom testiranju koda – prvo isprogramiraj funkcionalnost, a onda napiši testove idrugi pristup – TDD (Test Driven Development – ne smijemo kod percipirati kod s gledišta programera, napisati testove prije pisanja koda

- Ne moramo se fokusirati na to kako ćemo napisati kod nego na ono što taj kod treba raditi
- Napišite test
- Napišite minimalni kod koji prolazi taj test
- Refaktorirajte kod, kod se mora držati svih dobrih praksi



Problem jer se puno vremena gubi dok se napiše test, dok se osmisli test, test mora biti dobro koncipiran

Bitno je da testovi postoje, da je kod pokriven testovima, da bi kasnije imali manje problema

PHP nema integrirano rješenje
-treba nam dodatak PHP Unit test

https://phpunit.de/index.html

- Potebno je uz pomoć ovog paketa napraviti testove

```
composer require --dev phpunit/phpunit
```

Kod instalacije ga composer neće smjestiti uz require područje nego u require **dev, bitno je kada netko uzme u naš projekt ini što je urequire mora biti zadovoljeno, a ono što je u require dev nije obavezno**

1. Korak – što mi želino testirati
   - Ako znam kako router mora raditi, kako bi potencijalno ako se nešto promijeni test upozorio da nešto neće raditi

- Pokrenemo naš kod da nešto radi, a onda radimo assert – provjerava je li dobiveni rezultat – da li se taj koma dkoda ponaša u

Da li je vratio točno taj odgovor

Kako napisati test naprimjer za funkciju za zbrajanje;

Uzmemo funkciju, pošaljemo joj dva broja (kontrolirana dva broja) – da li je rezultat točan – test treba to potvrditi

Kako se funkcija ponaša ako joj pošaljemo string, ako u rezultatu ne dobijem očekivanu vrijednost, dobiti ćemo upozorenje, funkcija se ne ponaša u zadanom očekivanju

U testu se testira očekivano ponašanje, ne samo kad radi dobro nego kad i ne radi kako treba, upali li se try catch blok?

PHP UN naredbe

- assertTrue, ako je jednak true – test je uspješno proveden, ako je u assertTrue poslana vrijednost false iz nekog našeg koda – Failure

```php
<?php declare(strict_types=1);
final class Greeter
{
    public function greet(string $name): string
    {
        return 'Hello, ' . $name . '!';
    }
}
```

Example 2.2 A test class named `GreeterTest` (declared in `tests/GreeterTest.php`)

```php
<?php declare(strict_types=1);
use PHPUnit\Framework\TestCase;

final class GreeterTest extends TestCase
{
    public function testGreetsWithName(): void
    {
        $greeter = new Greeter;

        $greeting = $greeter->greet('Alice');

        $this->assertSame('Hello, Alice!', $greeting);
    }
}
```

Pišemo test scenarij za klasu Greeter. Instanciramo klasu i pošaljemo joj parametar 'Alice', provjeravamo funkcijom $this->**assertSame** je li očekivana vrijednost jednaka dobivenoj

Ph punit smo instalirali lokalno, a ne globalno, pa u git bashu moramo pokrenuti

```
4/NapredniPHP/AlgebraCRM (master)
$ ./vendor/bin/phpunit
```

Tetsovi se pišu tako da u rootu projekta dodamo novu mapu koja se zove tests,

Razdvajamo testove u **unit** i u **integration** (Laravel ih naziva Feature test)

3. Dodamo datoteku xyTest. Php, sva imena testova trebala bi završiti s **test.php**

```php
class RouterTest extends TestCase
{
    protected function setUp(): void
    {
        $_SERVER['REQUEST_METHOD'] = '';
        $_SERVER['REQUEST_URI'] = '';
    }
```

Test case dolazi iz PHP unita, svi naši testovi će se moći izvršiti na bazi TestCase

Svaki test može imati setup – prilikom setupa vršimo predefinirane elemente koje ćemo u testu koristiti (kao nekakav konstruktor) koji postavlja envirement za testiranmje

Bitno je također da sve metode počinju sa test

Testira je li router singleton.

```php
use App\Services\Router;
use PHPUnit\Framework\TestCase;

class RouterTest extends TestCase
{
    protected function setUp(): void
    {
        //
    }

    public function testGetInstance()
    {
        $router1 = Router::getInstance();
        $router2 = Router::getInstance();

        $this->assertSame($router1, $router2);
    }
}
```

Pokretanje testa:

```
$ ./vendor/bin/phpunit tests/unit/RouterTest.php
```

Unite testovi kažu da nešto ne radi, ali ne kažu gdje ne radi, zato se koriste integracijski testovi. Unit test ne daje potpune informacije.

Daljnje testiranje – kako simulirati podatke npr. $_SERVER['REQUEST_METHOD'] = 'GET';

$_SERVER['REQUEST_URI'] = APP_PATH . '/nonexist';

Trebamo napraviti mockove – set potrebnih informacija koje bi imali kada se stvarno dogodi nekakav request

Moramo npr. superglobalnu varijabli $_SERVER mokat – fejkat

To radimo u setupu testa

```php
protected function setUp(): void
{
    $_SERVER['REQUEST_METHOD'] = '';
    $_SERVER['REQUEST_URI'] = '';
}
```

Ako želimo testirati dispatcher treba nam Controller, pa ćemo mokati Controller (nemamo realini kontroler nego mock); također test očekuje konstatnu

```php
class TestController
{
    public function testAction()
    {
        echo 'Test Action';
    }
}
```

Ako se radi o unit testu onda to moramo staviti u mock, ne koristi vanjsku klasu – ne idemo po stvarne podatke, ne radimo s bazom

Kontinuirana integracija –

Kad se kod postavi na dev , izvrte se testovi i kad svi testovi prođu kaže sada deploy-a u stageing

UNIT TEST – testira se jedinica koja je okačena na ostale stvari, nema vezu s ostalim dijelom projekta

12. LARAVEL

Fasade – korištenje u Laravelu

Laravel generira htaccess

Autoloader

Diže boot strap

Diže servisni container

Instalacija:

composer cerate-project laravel/laravel ArgebraBlog

composer create-project laravel/laravel ime-projekta

php artisan serve

.env

SESSION_DRIVER=file

CACHE_STORE=file

Odkomentirati:

DB_CONNECTION=sqlite

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=laravel

DB_USERNAME=root

DB_PASSWORD=

DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=laravel-blog

DB_USERNAME=root

DB_PASSWORD=

Prvo pokrenemo:

-php artisan migrate

Ako trebamo promijeniti nešto u bazi, promjenimo u migracij

- Mormo naprviti rollback

Php atisan migrate:rollback – aktivira se metoda down u migraciji

Paziti kod rollbacka ako imamo podatke

Ponovo pokrenuti php artisan migrate

Dodavanje migracije za novu tablicu tablice:

php artisan make:migration create_roles_table

Create – kreira

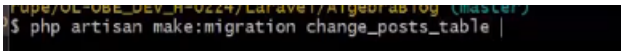
Roles – ime teblice

Table – znak da je to tablica

Kompozitni ključ za zavisne tabice (nema soft delete) – jedan korisnik može imati više rola, a jedna rola može pripadati više korisnika:

```
Schema::create('user_roles', function (Blueprint $table) {
    $table->foreignId('role_id')->constrained()->cascadeOnDelete();
    $table->foreignId('user_id')->constrained()->cascadeOnDelete();
    $table->primary(['role_id', 'user_id']);
    $table->timestamps();
});
```

$table->string('slug')->unique(); - kad radimo web stranice želimo biti što bolje pozicionirati, mora biti sale friendly url – automatski se generira iz naslova post-a



php artisan make:migration change_posts_table --table=posts – Migracija koja mijenja tablicu

```php
public function up(): void
{
    Schema::table('posts', function (Blueprint $table) {
        //
        $table->softDeletes();
    });
}
```

MODELI

Kad kreiramo modele moraju biti eloquent modeli, svaki naš model mora naslijediti odnosno proširiti se s modelom koji dolazi iz elogventa

Ime modela mora biti jednina – na taj način eloquent mapira model s tablicom u bazi

Tablica plural, model singular

Ne nasljeđuje model, ali Authenticatable je klasa – kako bi imali nad njime mehanizam za autentifikaciju i autorizaciju korisnika

Obavezno model mora User uvijek mora naslijediti Usera (koji je alias da se ne sudari s našim) i to se ne dira.

use Illuminate\Foundation\Auth\User as Authenticatable;

use Illuminate\Notifications\Notifiable;

class User extends Authenticatable

Factory se koristi nad modelom da bi mogli raditi Seaderi

Kako bi mogli koristiti Seaderi

class User extends Authenticatable

{

use HasFactory, Notifiable; - obavezno koristiti

Kreiranje modela:

```
# Generate a model and a migration, factory, seeder, and controller...
php artisan make:model Flight -mfsc

# Shortcut to generate a model, migration, factory, seeder, policy, controller, 
php artisan make:model Flight --all
php artisan make:model Flight -a
```

https://laravel.com/docs/11.x/eloquent

php artisan make:model Role

https://laravelmagazine.com/differences-between-lazy-loading-and-eager-loading

DEFINIRANJE RELACIJA

Pogledati u php user, post i role i tamo (one to many, belongs to ...)

Sederi – Pogledati Factory i koja poalja su obavezna UserFactory.php – pomoću fakera

php artisan db:seed – pokreće glavni datoteku DatabaseSeeder

CONTROLERI

HTTP – Controlers

Defaut klasa Controller.php

Laravel generira controlere taj tip kontrolera može utjecati na bazic ,etode

ResourceControllers – nudi predefinirane CRUD operacije i nudi rutu – Resource rute i on u njoj raspozna različite requestove i zna ih rasporediti na isprvnu metodu u Controleru

Imali smo popis ruta gdje smo za svaku pojedinu rutu definirali neku akciju

php artisan make:controller AuthController

Osim Controllera trebaju nam i rute, ruta prosljeđuje zhtjev na Controller

Ruta prosljeđuje zahtjev na controller

Web.php

Ima mogućnost da u samoj ruti pošaljemo callback funkciju koja izvrši ono što

Smo u njoj napisali – znači zahtjev se može poslati callback funkciji, a može se poslati controlleru

Parametar – controller i metodu šaljem u arrey

php artisan make:controller UserController

Prvi argument je naziv controllera, a drugi je metoda login