**Hyper-V virtualka (radno okruženje)**

* login na fizičko računalo ispred vas

user: STUDENT

pass: STUDENT

* upute za sve u pdf obliku u Downloads (prebacit će profesor sa mreže)
* na računalu će biti 2 Hyper-V virtualke (Ubuntu Server i Windows virtualka Ispit) sa loginom:

user: algebra

pass: Pa$$w0rd

* kopirati ili prepisati pitanje iz testne aplikacije u VS Code pa potom provjeriti sa AI kroz web browser

// označiti cijeli tekst pa copy/paste

CTRL + A

CTRL + C

CTRL + V

// chatgpt

https://chatgpt.com/

TODO (sami instalirat Ubuntu na Hyper-V i shvatit kako se snaći u tom virtualnom okruženju)

**Spajanje xampp i vscode**

* downloadajte najnoviji XAMPP i instalirajte ga (ako već ne postoji na sustavu)
* na localdisk C: pronađite xampp, i u njemu php direktorij te kopirajte adresu putanje (desni klik na php i copy address)
* adresu (koja bi trebala ovako izgledati - C:\xampp\php) spremite u system environment path varijablu vaših Windowsa (upute kako doći do nje na linku)

https://www.eukhost.com/kb/how-to-add-to-the-path-on-windows-10-and-windows-11/

* nakon toga otvorite vaš VSCode, kliknite na Open Folder i dođite putanjom C:\xampp\htdocs do htdocs foldera u kojem kreirate direktorij za vaš projekt (nakon toga možete stvoriti .php stranicu i krenuti sa kodiranjem)
* rezultate možete provjeriti otvaranjem localhost stranice u web browseru (localhost/ime\_stranice ako ih imate više)
* preporuka instalirati PHP intelephense ekstenziju ako već nije

**Document root (korijenska mapa)**

- document root (korijenska mapa) odnosi se na direktorij najvišeg nivoa gdje web server poslužuje datoteke za određenu web stranicu ili aplikaciju

- za web servere na Linux sustavima obično je postavljen na /var/www/html

**Spajanje na bazu (i dohvat podataka) pomoću Mysqli funkcije - proceduralno**

$hostname = localhost;

$username = algebra;

$password = algebra;

$database = videoteka;

// opcionalno

$port = 3306;

// stvaranje konekcije na bazu

$connection = mysqli\_connect($hostname, $username, $password, $database, $port);

// provjera je li sve prošlo bez grešaka

if (mysqli\_connect\_errno()) {

die("Pogreška kod spajanja na poslužitelj: " . mysqli\_connect\_error());

}

echo "Spojeni ste na bazu";

// zatvaranje konekcije

mysqli\_close($connection);

// query i podaci

$query = 'SELECT \* FROM users WHERE city = ? ORDER BY name';

$param = 'Zagreb';

// funkcija za pripremu SQL naredbe, sigurno vezanje parametara, te izvođenje naredbe, sve u jednom

$result = mysqli\_execute\_query($connection, $query, $param);

**Spajanje na bazu (i dohvat podataka) pomoću PDO klase**

// potrebni podaci

$host = 'localhost';

$database = 'videoteka';

$username = 'algebra';

$password = 'algebra';

// opcionalno

$port = 3306;

$charset = 'utf8mb4';

// dodatne opcije, kako će se dohvaćati podaci (fetch - associjativno polje) i kako će se prikazivati greške (error mode - exception)

$options = [PDO::ATTR\_DEFAULT\_FETCH\_MODE => PDO::FETCH\_ASSOC, PDO::ATTR\_ERRMODE => PDO::ERRMODE\_EXCEPTION];

// dsn (data source name) - osnovni podaci o bazi, tip (u ovom slučaju mysql), host, naziv i (opcionalno) port, charset

$dsn = "mysql:host={$host};port={$port};dbname={$database};charset={$charset}"

// konekcija sa dsn, username, password i dodatnim opcijama (dohvat podataka, greške i slično)

$pdo = new PDO($dsn, $username, $password, $options);

// zatvaranje konekcije (u biti nuliranje varijable koja sadrži konekciju na bazu)

$pdo = NULL;

// query i podaci

$query = 'SELECT \* FROM users WHERE city = ? ORDER BY name';

$param = 'Zagreb';

// metode za pripremu i izvođenje naredbe

$statement = $pdo->prepare($sql);

$statement->execute($param);

// metode za dohvat podataka (fetch dohvati samo jedan redak, fetchAll dohvati sve)

$result = $statement->fetch();

$result = $statement->fetchAll();

**HTML forma**

* napraviti formu koja će slati username i password sa POST metodom (potreban je i submit button)

<?php

<form action="/login" method="POST">

<div>

<label for="username">Username</label>

<input type="text" id="username" name="username" required>

</div>

<div>

<label for="password">Password</label>

<input type="password" id="password" name="password" required>

</div>

<hr>

<div>

<button type="submit">Login</button>

</div>

</form>

* primjer unosa filea (u form treba dodati enctype="multipart/form-data" te type="file")

<?php

<form action="/data" method="POST" enctype="multipart/form-data">

<div>

<label for="file">Datoteka</label>

<input type="file" id="file" name="file" required>

</div>

<div>

<button type="submit">Upload</button>

</div>

</form>

* primjer update forme (za delete umjesto PATCH/PUT stavite DELETE)

<?php

<form action="/update" method="POST">

<input type="hidden" name="\_method" value="PATCH">

<input type="hidden" name="id" value="<?= $user['id'] ?>">

<div>

<label for="username">Username</label>

<input type="text" id="username" name="username" value="$user['username']" required>

</div>

<div>

<label for="old\_password">Old password</label>

<input type="password"id="old\_password" name="old\_password" required>

</div>

<div>

<label for="new\_password">New password</label>

<input type="password"id="new\_password" name="new\_password" required>

</div>

<div>

<label for="confirm\_password">Confirm password</label>

<input type="password"id="confirm\_password" name="confirm\_password" required>

</div>

<hr>

<div>

<button type="submit">Update</button>

</div>

</form>

**Sortiranje**

// funkcije za sortiranje nizova:

sort() – indeksirani nizovi, rastući redoslijed (ASC)

rsort() – indeksirani nizovi, padajući redoslijed (DESC)

asort() – asocijativni nizovi, rastući redoslijed (ASC), čuva key-value parove

arsort() – asocijativni nizovi, opadajući redoslijed (DESC), čuva key-value parove

ksort() – asocijativni nizovi, rastući redoslijed prema ključevima (ASC)

krsort() – asocijativni nizovi, opadajući redoslijed prema ključevima (DESC)

// sortiranje sa uključenom anonimnom funkcijom:

usort() – indeksirani nizovi

uasort() – Korisničko sortiranje (čuva ključeve, asocijativni nizovi)

uksort() – Korisničko sortiranje prema ključevima (asocijativni nizovi)

// sortiranje više nizova:

array\_multisort() – Sortiranje više nizova ili više-dimenzionalnog niza

// prirodno sortiranje (znači da je primjerice a10 iza a2)

natsort() – ne osjetljivo na velika i mala slova

natcasesort() – osjetljivo na velika i mala slova

**.txt .csv .json datoteke**

Kreiranje direktorija, otvaranje, čitanje i pisanje u datoteke

// podaci

$data = [

['Name', 'Age', 'City'],

['John Doe', 30, 'New York'],

['Jane Smith', 25, 'Los Angeles']

];

// postavljanje putanje direktorija

$dir = 'putanja/do/direktorija';

// kreiranje direktorija

if (!is\_dir($dir)) {

mkdir($dir, 0777, true); // ako direktorij ne postoji, kreira ga s dozvolama 0777

}

// putanje do datoteka

$txtFile = $dir . '/file.txt';

$csvFile = $dir . '/file.csv';

$jsonFile = $dir . '/file.json';

// TXT

// čitanje cijele .txt datoteke

if (file\_exists($txtFile)) {

$txtContent = file\_get\_contents($txtFile); // učitava cijeli sadržaj u varijablu

echo "Content of file.txt:\n$txtContent\n";

} else {

exit("file.txt does not exist"); // ako datoteka ne postoji, izlazi iz skripte

}

// čitanje .txt datoteke redak po redak

$txtFileHandle = fopen($txtFile, 'r'); // otvara datoteku u modu za čitanje

if ($txtFileHandle === false) {

exit("Unable to open file for reading"); // ako nije moguće otvoriti datoteku, izlazi iz skripte

}

while (($txtRow = fgets($txtFileHandle)) !== false) {

echo "TXT Row: " . $txtRow . "\n"; // ispisuje svaki redak iz datoteke

}

fclose($txtFileHandle); // Zatvaramo datoteku

// pisanje u .txt datoteku redak po redak

$file = fopen($txtFile, 'a'); // otvara datoteku u "append" modu za dodavanje novih podataka

if ($file === false) {

exit("Unable to open file for writing"); // ako nije moguće otvoriti datoteku za pisanje, izlazi iz skripte

}

foreach ($data as $row) {

fwrite($file, implode(", ", $row) . "\n"); // pretvara svaki redak u string i zapisuje u datoteku

}

fclose($file); // zatvaramo datoteku

// CSV

// čitanje .csv datoteke

if (file\_exists($csvFile)) {

$csvFileHandle = fopen($csvFile, 'r'); // otvara csv datoteku u modu za čitanje

if ($csvFileHandle === false) {

exit("Unable to open file for reading"); // ako nije moguće otvoriti datoteku, izlazi iz skripte

}

$csvHeader = fgetcsv($csvFileHandle); // čita zaglavlje csv datoteke

echo "CSV Header: " . implode(", ", $csvHeader) . "\n"; // ispisuje zaglavlje

while (($csvRow = fgetcsv($csvFileHandle)) !== false) { // čita svaki redak csv datoteke

echo "CSV Row: " . implode(", ", $csvRow) . "\n"; // ispisuje redak

}

fclose($csvFileHandle); // zatvara datoteku

} else {

exit("file.csv does not exist"); // ako csv datoteka ne postoji, izlazi iz skripte

}

// pisanje u .csv datoteku

$file = fopen($csvFile, 'a'); // otvara datoteku u "append" modu za dodavanje novih podataka

if ($file === false) {

exit("Unable to open file for writing"); // ako nije moguće otvoriti datoteku za pisanje, izlazi iz skripte

}

foreach ($data as $row) {

fputcsv($file, $row); // zapisuje redak u csv datoteku

}

fclose($file); // zatvara datoteku

// JSON FILE

// čitanje i dekodiranje .json datoteke

if (file\_exists($jsonFile)) {

$jsonContent = file\_get\_contents($jsonFile); // učitava cijeli sadržaj json datoteke

if ($jsonContent === false) {

exit("Unable to open file for reading"); // ako nije moguće otvoriti datoteku, izlazi iz skripte

}

$jsonData = json\_decode($jsonContent, true); // dekodira json u asocijativno polje

if ($jsonData === false) {

exit("Unable to decode file for reading"); // ako dođe do greške pri dekodiranju json-a, izlazi iz skripte

}

print\_r($jsonData); // ispisuje sadržaj dekodiranog json-a

} else {

exit("file.json does not exist"); // ako json datoteka ne postoji, izlazi iz skripte

}

// enkodiranje i pisanje u .json datoteku

$jsonData = json\_encode($data, JSON\_PRETTY\_PRINT); // pretvara podatke u json format, sa "pretty print" formatiranjem

if ($jsonData === false) {

exit("Unable to encode data for writing"); // ako dođe do greške pri enkodiranju, izlazi iz skripte

}

// dodavanje novih podataka u json datoteku

if (file\_put\_contents($jsonFile, $jsonData, FILE\_APPEND) === false) { // zapisuje nove podatke na kraj json datoteke

exit("Unable to open file for writing"); // ako nije moguće zapisati podatke u datoteku, izlazi iz skripte

}

**PHP sesije ($\_SESSION)**

Sesije omogućuju pohranu podataka između različitih stranica i zahtjeva, koriste se za pohranu podataka o korisnicima, preferencije i druge informacije koje želite pratiti dok korisnik navigira kroz vašu web stranicu.

// pokretanje sesije

session\_start();

// regeneracija ID-a (poboljšava sigurnost, primjerice koristimo je nakon logina)

session\_regenerate\_id();

// zatvaranje sesije

session\_unset(); // uklanja podatke sesije

session\_destroy(); // uništava samu sesiju

**PHP tipovi podataka**

skalarni tipovi podataka - brojčani (int, float), tekstualni (string), boolean (true ili false)

složeni tipovi podataka - polja/nizovi (indeksirana, asocijativna), objekti (instance klasa, kolekcije)

specijalni tipovi - NULL, resource (primjerice konekcije na baze podataka ili datoteke)

**Varijable**

Varijabla je imenovana memorijska lokacija koja sadrži podatke kojima je moguće manipulirati izvođenjem programa. Lokalne varijable su deklarirane u funkciji, globalnim varijablama nije moguće direktno pristupiti unutar funkcije (mora im se dodati GLOBAL ili ih predati prilikom poziva funkcije), statične varijable (oznaka STATIC) zadržavaju svoju vrijednost i nakon izlaska iz funkcije, superglobalne varijable su dostupne bilo gdje unutar skripte (primjer $\_SERVER).

**Poziv po referenci**

Vrijednosti se funkciji mogu proslijediti kao statične vrijednosti (direktan upis neke vrijednosti, odnosno brojke prilikom poziva funkcije), kao vrijednosti iz varijabli (poziv po vrijednosti) ili po referenci. Poziv po referenci omogućava funkciji da izmijeni vrijednost varijable koja je proslijeđena, umjesto da radi s kopijom te varijable. Kada varijablu proslijedite po referenci, promjene koje se naprave u funkciji odražavaju se i izvan funkcije.

<?php

// &$num - & (adresni operator - vraća memorijsku lokaciju varijable) ispred $num znači da se $num prosljeđuje po referenci, a ne po vrijednosti

function addTen(&$num) {

$num += 10; // ovo će modificirati originalnu varijablu

}

$number = 5;

echo $number; // prije poziva funkcije ispisuje 5

addTen($number);

echo $number; // nakon poziva funkcije ispisuje 15

Poziv po vrijednosti - funkcija radi s kopijom varijable i sve promjene unutar funkcije ne utječu na originalnu varijablu. Poziv po referenci - funkcija radi izravno na stvarnoj varijabli i promjene utječu na varijablu i izvan funkcije.

**Programske petlje (foreach, for, while)**

- programske petlje su strukture koje omogućavaju da se dijelovi programa/koda izvrše, odnosno iteriraju više puta (zadani broj ili sve dok je određeni uvjet ispunjen), te na taj način ubrzavaju/automatiziraju obradu podataka (izbjegavamo repetitivno ponavljanje koda), posebno su korisne kada treba obaviti akciju nad nizom elemenata (primjerice prilikom pretraživanja lista, polja i slično).

**do-while petlja**

<?php

$i = 0;

do {

echo $i . "\n";

$i++;

} while ($i <= 77);

**spl\_autoload\_register**

Funkcija spl\_autoload\_register() pojednostavljuje proces uključivanja datoteka klasa u PHP-u, ona se koristi za definiranje i registraciju autoload funkcije, te zajedno sa njom čini mehanizam koji automatski učitava PHP klase kada su potrebne, bez potrebe za ručnim uključivanjem ili zahtijevanjem datoteka s klasama.

spl\_autoload\_register(function ($class) {

// u tijelu anonimne autoload funkcije definiramo kako učitati datoteku klase (u ovom slučaju iz direktorija classes)

require\_once 'classes/' . $class . '.php';

});

**OOP $this-> i self::**

$this - odnosi se na trenutnu instancu klase (objekt), koristi se za pristupanje svojstvima i metodama koje nisu označene kao static

-> - operator koji se koristi za pristup svojstvima i metodama objekta instance

self - odnosi se na trenutnu klasu (ne na instancu klase/objekt) i koristi se za pristupanje statičkim svojstvima i metodama

:: - operator rezolucije opsega (scope operator) koji se koristi za pristupanje statičkim metodama/svojstvima i konstantama, ili za referenciranje same klase

**Pretvaranje funkcije zbroj u klasu**

* u klasi kroz konstruktor definiramo dvije privatne varijable, numberA i numberB, dvije public metode za dohvaćanje/promjenu njihovih vrijednosti (settere), te public funkciju za zbrajanje

function sum(int $a, int $b): int

{

return $a + $b;

}

class Sum

{

public function \_\_construct(

private int $numberA = 0,

private int $numberB = 0,

) {}

public function setA(int $a)

{

$this->numberA = $a;

}

public function setB(int $b)

{

$this->numberB = $b;

}

public function sum(?int $a = null, ?int $b = null ): int

{

return ($a ?? $this->numberA) + ($b ?? $this->numberB);

}

}

$zbroj = new Sum();

$zbroj->sum(); // vratit će 0

$zbroj->setA(7);

$zbroj->setB(10);

$zbroj->sum(); // vratit će 17

$zbroj->sum(4, 5); // vratit će 9

**Klasa kalkulator**

<?php

class Calculator

{

// property promotion - deklariramo svojstva (varijable) i ujedno im pridjeljujemo vrijednost

public function \_\_construct(

private float $a,

private string $operator,

private float $b,

) {}

// metoda koja poziva jednu od metoda za izračun ovisno o unesenom operatoru

public function calculate(): float

{

return match ($this->operator) {

'+' => $this->add(),

'-' => $this->subtract(),

'\*' => $this->multiply(),

'/' => $this->divide(),

default => throw new InvalidArgumentException("unesen nepostojeći operator za izračun"),

};

}

private function add(): float

{

return $this->a + $this->b;

}

private function subtract(): float

{

return $this->a - $this->b;

}

private function multiply(): float

{

return $this->a \* $this->b;

}

private function divide(): float

{

if ($this->b == 0) {

throw new InvalidArgumentException("nemože se dijeliti sa nulom");

}

return $this->a / $this->b;

}

}

// primjer korištenja

try {

$calc1 = new Calculator(10, '+', 5);

echo "Zbroj: " . $calc1->calculate() . "\n";

$calc2 = new Calculator(10, '-', 5);

echo "Oduzimanje: " . $calc2->calculate() . "\n";

$calc3 = new Calculator(10, '\*', 5);

echo "Množenje: " . $calc3->calculate() . "\n";

$calc4 = new Calculator(10, '/', 5);

echo "Dijeljenje: " . $calc4->calculate() . "\n";

// primjer dijeljenja sa nulom

$calc5 = new Calculator(10, '/', 0);

echo "Dijeljenje sa nulom: " . $calc5->calculate() . "\n";

} catch (Exception $e) {

echo "Greška - " . $e->getMessage() . "\n";

}

?>

**array\_map()**

* funkcija array\_map() kao prvi argument može primiti callback ili anonimnu funkciju, a kao ostale argumente prima jedno ili više polja vrijednosti (koje onda redom koristi u funkciji danoj sa prvim argumentom)

// array\_map prima callback funkciju

function squares($n)

{

return ($n \* $n);

}

$squares = array\_map('squares', [2, 3, 4, 5, 6]);

// array\_map prima anonimnu funkciju (može i skraćeni zapis, array funkcija - fn($n) => return $n\*$n)

$squares = array\_map(function($n) {return ($n \* $n);}, [2, 3, 4, 5, 6]);

// kombinacija callback i anonimne (array) funkcije

$squares = fn($n) => return $n\*$n;

$squares = array\_map($squares, [2, 3, 4, 5, 6]);

// rezultat je u sva 3 slučaja isti

print\_r($squares);

array(

[0] => 4

[1] => 9

[2] => 16

[3] => 25

[4] => 36

)

https://www.php.net/manual/en/function.array-map.php

**Git naredbe**

// inicijalizacija

git init

// postavljanje username i email u git konfiguracijsku datoteku (global označava da se postavka primjenjuje za sve git repozitorije na vašem računalu)

git config --global user.name ime\_korisnika

git config --global user.email email\_korisnika

// prikaz podataka iz git konfiguracijske datoteke

git config --global --list

// postupak spajanja sa udaljenim repozitorijem

// postavljanje promjena koje želimo poslati na udaljeni repozitorij, . označava da postavljamo sve, a možemo dodavati i zasebno fileove ili foldere

git add .

// svakom commitu (odnosno slanju podataka) se mora dodati poruka

git commit -m "prvi commit"

// označavamo udaljeni repozitorij kao origin

git remote add origin adresa\_repozitorija

// šaljemo sa master grane lokalnog repozitorija na udaljeni origin repozitorij, -u (set upstream) konfigurira lokalnu granu da prati udaljeni repozitorij

git push -u origin master

// kopiranje projekta sa nekog repozitorija

git clone repo\_path

// dohvat te spajanje podataka sa udaljenog repozitorija

git fetch

git merge

// skraćeni način, automatski napravi i dohvat, i spajanje

git pull

// skraćeni način slanja podataka sa lokalne grane na udaljeni repozitorij, nakon što smo prilikom prvog pusha sve konfigurirali (origin i set upstream)

git push

// trenutno stanje gita, grana, promjene, itd.

git status

// undo svih promjena ako nismo napravili add-commit, . sve promjene, a mogu se i zasebno navesti pojedini folderi i fileovi

git checkout .

// undo add

git reset

// undo samo commita

git reset --soft HEAD^

// undo commit i add

git reset HEAD^

// undo commit, add i svih promjena

git reset --hard HEAD^

// ispis grane na kojoj smo trenutno

git branch

// kreiranje nove grane lokalno, ali ne i automatski prijelaz na nju

git branch nova

// prijelaz na novu granu

git checkout nova

// kreiranje nove grane i automatski prijelaz na nju u jednoj naredbi (-b oznaka je ključna)

git checkout -b nova

// stavljanje novostvorene grane na udaljeni repozitorij

git push -u origin nova

// dohvaćanje najnovije verzije nove grane sa udaljenog repozitorija

git fetch

git merge origin nova

// uzastopno odrađene fetch i merge naredbe

git pull origin nova

// prebacivanje na master granu

git checkout master

// dohvat filea iz nove grane u master

git checkout nova ime\_filea

// spajanje svih podataka iz nove grane u master

git merge nova

// brisanje nove grane na udaljenom repozitoriju

git push -d origin nova

// brisanje nove grane lokalno, prije toga se sa checkout morate premjestiti na neku drugu granu

git branch -d nova

**Prikaz, kreiranje, brisanje direktorija i datoteka u Linux terminalu**

// prikaz svih datoteka i poddirektorija unutar navedenog direktorija, -al određuje prikaz svih elemenata (i onih skrivenih - a kao all), i to poredanih u listu sa prikazom ovlasti (l kao list)

ls -al ime\_direktorija

// ll je alias za ls -al spremljen u .bashrc

ll ime\_direktorija

// kreiranje direktorija

mkdir ime\_direktorija

// brisanje praznog! direktorija

rmdir ime\_direktorija

// kreiranje nove datoteke

touch ime\_datoteke

// brisanje direktorija i datoteka, možemo ih navesti više jedno iza drugog, ponekad potrebne sudo ovlasti (-r oznaka da obriše sve poddirektorije i datoteke unutar navedenog direktorija)

rm -r ime\_direktorija ime\_datoteke

// ispis u Linux terminalu

cat ime\_datoteke

// otvaranje/prikaz datoteke sa posebnim programom, datoteka se ako ne postoji automatski stvori, potrebna sudo ovlast

nano ime\_datoteke

vim ime\_datoteke

**Zapisivanje u file kroz Linux terminal**

// sa operatorom >> i naredbom echo dodajemo tekst kao novu liniju na kraj datoteke

echo "Hello World" >> file.txt

// sa operatorom > dodajemo tekst i brišemo stari sadržaj datoteke ako postoji

echo "Hello World" > file.txt

* dodatne opcije

https://www.baeldung.com/linux/file-append-text-no-redirection

**Provjera PHP verzije u Linux terminalu**

php -v

php --version

// ispis cijelog phpinfo() filea

php -i

php --info

// prikaz putanja svih php konfiguracijskih datoteka

php --ini

**Ostale Linux terminal naredbe**

// prikaz patha direktorija u kojem se trenutno nalazimo

pwd

// promjena direktorija

cd path

// prijelaz u direktorij koji se nalazi unutar trenutnog direktorija

cd ime\_direktorija

// prijelaz na nivo iznad trenutnog direktorija

cd ..

// povratak nivo iznad te ulazak u neki drugi poddirektorij

cd ../ime\_direktorija

// povratak u home direktorij korisnika

cd ~

// prikaz opisa naredbe

man ime\_naredbe

ime\_naredbe --help

// kopiranje datoteke

cp stara\_datoteka neki\_direktorij/nova\_datoteka

// preimenovanje datoteke (ako je ishodište i odredište u istome direktoriju)

mv staro\_ime novo\_ime

// premještanje datoteke (ako je odredište u nekom drugom direktoriju, a moguće joj je dati i novo ime)

mv stara\_datoteka drugi\_direktorij/nova\_datoteka

// super user do - oznaka da naredbu izvodimo sa root privilegijama

sudo ime\_naredbe

// prikaz veličine datoteke/direktorija (oznakom -m prikazujemo veličinu u megabajtima)

du -m ime\_datoteke

// kompresija/dekompresija datoteka/direktorija

zip ime\_direktorija

unzip ime\_direktorija

// dohvat updatea i upgrade nekog paketa ili svih programa Linux sustava (ako ne navedemo određeni paket), oznakom -y odgovaramo potvrdno (yes) na sve upite tokom instalacije

sudo apt-get update ime\_paketa

sudo apt-get upgrade -y ime\_paketa

// promjena privilegija (read 2^2, write 2^1, execute 2^0) nad određenom datotekom/direktorijem, -R oznakom (recursive) mijenjamo vlasništvo nad fileovima, te poddirektorijima koji se nalaze unutar navedenog direktorija

chmod -R 777 ime\_datoteke

// primjer dodavanja privilegija sa User/Group/Other i Read/Write/eXecute

chmod u+rwx, g+rx, o+r ime\_datoteke

// oduzimanje read i execute privilegija grupi nad određenom datotekom

chmod g-rx ime\_datoteke

// davanje privilegija read i write svima, user, group i other (oznaka a - all)

chmod a+rw ime\_datoteke

// promjena vlasništva (user:group) nad datotekom/direktorijem, -R oznakom mijenjamo vlasništvo nad fileovima, te poddirektorijima koji se nalaze unutar navedenog direktorija

chown -R algebra:algebra ime\_direktorija

// prikaz ip adrese

hostname -I

// provjera konekcije prema navedenoj ip adresom

ping neki\_ip

// prikaz id trenutnog korisnika

echo $UID

// brisanje zaslona terminala

clear

// zaustavljanje izvođenja naredbe u terminalu

CTRL+C

// prisilno zaustavljanje izvođenja naredbe u terminalu

CTRL+Z

// izlazak iz terminala

exit

**Git/Linux zadatak**

1. Kreirajte datoteku `app.php` i dodajte u nju liniju teksta sa echo naredbom:

touch app.php

echo "pozdrav" > app.php

2. Inicijalizirajte Git repozitorij i dodajte datoteku u staging area:

git init

git add app.php

3. Napravite commit sa porukom "My first commit":

git commit -m "My first commit"

4. Kreirajte novu granu feature-remove-echo i prebacite se na nju:

git checkout -b feature-remove-echo # kreiranje i prebacivanje u jednom sa oznakom -b (inače bi morali napraviti git branch za kreiranje, pa git checkout za prebacivanje)

5. Dodajte još jednu echo liniju u app.php i napravite commit:

echo "pozdrav ponovno" >> app.php # dodavanje na kraj filea (sa > bi prebrisali prijašnji sadržaj)

git add app.php

git commit -m "Added second echo line"

6. Sjedinite (merge) granu feature-remove-echo u master:

git checkout master # prebacivanje na master

git merge feature-remove-echo # merge u master granu

**MySQL relacije (1-1, 1-n, n-m)**

| **Odnos** | **Opis** | **Implementacija** |
| --- | --- | --- |
| 1-1 | Jedan zapis u tablici A odnosi se na jedan zapis u tablici B. | Dodajte strani ključ u jednu tablicu koji referencira primarni ključ druge tablice. |
| 1-n | Jedan zapis u tablici A odnosi se na mnoge zapise u tablici B. | Dodajte strani ključ u "n" tablicu koji referencira primarni ključ "1" tablice. |
| n-m | Mnogi zapisi u tablici A odnose se na mnoge zapise u tablici B. | Kreirajte spojnu (pivot) tablicu sa stranim ključevima koji referenciraju obje tablice. |

**Normalizacija**

MySQL normalizacija odnosi se na proces organiziranja podataka u relacijskim bazama kako bi se smanjila redundantnost i poboljšao integritet podataka te učinkovitosti upita.

To uključuje razbijanje velikih, složenih tablica na manje, jednostavnije te uspostavljanje odnosa (zavisnosti, strani ključevi) između njih.

Normalne forme

1NF - osiguranje atomskih vrijednosti (nema više od jednog predmeta u jednom stupcu)

2NF - osiguranje da nema parcijalnih zavisnosti (npr. atributa koji ovise samo o dijelu kompozitnog ključa)

3NF - osiguranje da nema tranzitivnih zavisnosti (npr. neključni atributi koji ovise o drugim neključnim atributima)

**SQL pretvaranje entiteta u relacije**

* zaposlenik može tokom vremena odraditi više poslova, a na svakom poslu može raditi više zaposlenika

ZAPOSLENIK(id, ime, prezime, adresa)

ODRAĐENI\_POSLOVI (id\_zaposlenik, id\_posao, datum)

POSAO (id, naziv)

// odrađeni\_poslovi je pivot tablica između zaposlenika i posla

ZAPOSLENIK 1 - n ODRAĐENI\_POSLOVI n - 1 POSAO

**Primjer ograničavanja korisnika na samo čitanje iz baze podataka**

-- kreiranje korisnika

CREATE USER 'user'@'localhost' IDENTIFIED BY 'password';

-- dodjela privilegija za samo čitanje

GRANT SELECT ON database.\* TO 'user'@'localhost';

-- opcionalno - oduzimanje drugih privilegija

REVOKE INSERT, UPDATE, DELETE ON database.\* FROM 'user'@'localhost';

-- primjena/reset promjena

FLUSH PRIVILEGES;

-- provjera privilegija

SHOW GRANTS FOR 'user'@'localhost';

**SQL procedura za izmjenu količine**

* mini tutorial za mysql procedure

https://www.dolthub.com/blog/2024-01-17-writing-mysql-procedures/

* napraviti proceduru koja će u tablici proizvodi mijenjati količine ovisno o količini prodanih proizvoda

DROP DATABASE IF EXISTS `pekara`;

CREATE DATABASE IF NOT EXISTS `pekara` DEFAULT CHARACTER SET utf8mb4 COLLATE utf8mb4\_general\_ci;

USE `pekara`;

CREATE TABLE IF NOT EXISTS `proizvodi` (

`id` INT UNSIGNED NOT NULL AUTO\_INCREMENT,

`naziv` VARCHAR(100) COLLATE utf8mb4\_general\_ci NOT NULL,

`kolicina` INT UNSIGNED NOT NULL

);

INSERT INTO `proizvodi` (`naziv`, `kolicina`) VALUES

('kruh', '1000'),

('pecivo', '500'),

('burek', '200'),

('buhtla', '200'),

('sendvic', '100');

-- procedura za izmjenu količine

DROP PROCEDURE IF EXISTS `izmjena\_kolicine`;

DELIMITER //

CREATE PROCEDURE IF NOT EXISTS `izmjena\_kolicine`(

IN prodan\_proizvod\_id INT UNSIGNED,

IN prodana\_kolicina INT UNSIGNED

)

BEGIN

DECLARE stara\_kolicina INT UNSIGNED;

START TRANSACTION;

-- provjera postoji li dovoljna količina

IF prodana\_kolicina <= 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Prodana količina mora biti veća od nule';

END IF;

-- provjera postoji li proizvod

SELECT kolicina INTO stara\_kolicina

FROM proizvodi

WHERE id = prodan\_proizvod\_id

FOR UPDATE; -- zaključavamo element dok se ne izvrši transakcija

-- ako ne postoji

IF stara\_kolicina IS NULL THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Proizvod s tim ID-om ne postoji';

END IF;

-- provjera i ažuriranje količine

IF (stara\_kolicina - prodana\_kolicina) >= 0 THEN

UPDATE proizvodi

SET kolicina = (stara\_kolicina - prodana\_kolicina)

WHERE id = prodan\_proizvod\_id;

-- slanje promjena

COMMIT;

ELSE

-- ako nema dovoljno proizvoda

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Nema dovoljno proizvoda na skladištu';

END IF;

END //

-- poziv procedure

CALL izmjena\_kolicine(2, 5);

**SQL transakcija, procedura i funkcija za prijenos i ispis količine**

-- kreiramo novu bazu podataka

DROP DATABASE IF EXISTS banka;

CREATE DATABASE IF NOT EXISTS banka;

USE banka;

-- kreiramo tablicu račun

CREATE TABLE IF NOT EXISTS accounts (

id INT UNSIGNED NOT NULL AUTO\_INCREMENT PRIMARY KEY,

iban VARCHAR(34) UNIQUE NOT NULL,

balance DECIMAL(15, 2) DEFAULT 0.00,

);

-- primjer transakcije za prebacivanje iznosa iz jednog računa na drugi

START TRANSACTION;

-- provjera ima li na računu dovoljno sredstava za plaćanje

SELECT balance INTO sender\_balance

FROM accounts

WHERE iban = sender\_iban;

-- ako ima dovoljno sredstava pokreni transakciju

IF sender\_balance >= transfer\_amount THEN

-- oduzmi sumu iz računa pošiljatelja

UPDATE accounts

SET balance = balance - transfer\_amount

WHERE iban = sender\_iban;

-- dodaj sumu na račun primatelja

UPDATE accounts

SET balance = balance + transfer\_amount

WHERE iban = receiver\_iban;

-- provjera jesu li oba ažuriranja prošla uspješno

IF ROW\_COUNT() = 2 THEN

COMMIT; -- ako je napravimo commit

ELSE

ROLLBACK; -- inače povratak na početno stanje

END IF;

ELSE

-- ako nema dovoljno sredstava na računu vrati na početno stanje

ROLLBACK;

END IF;

-- primjer procedure za prebacivanje iznosa iz jednog računa na drugi

DELIMITER $$

CREATE PROCEDURE make\_transaction(

IN sender\_iban VARCHAR(34),

IN receiver\_iban VARCHAR(34),

IN transfer\_amount DECIMAL(15, 2)

)

BEGIN

DECLARE sender\_balance DECIMAL(15, 2);

DECLARE new\_sender\_balance DECIMAL(15, 2);

DECLARE receiver\_balance DECIMAL(15, 2);

DECLARE new\_receiver\_balance DECIMAL(15, 2);

START TRANSACTION;

IF transfer\_amount <= 0 THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Količina mora biti veća od nule';

END IF;

-- dohvati stanje na računu pošiljatelja

SELECT balance INTO sender\_balance

FROM accounts

WHERE iban = sender\_iban;

FOR UPDATE; -- ovime eliminiramo race condition, odnosno mogućnost da netko promjeni stanje prije no što mi obavimo update

-- provjera je li dohvat prošao uspješno

IF sender\_balance IS NULL THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Pošiljatelj nije pronađen';

END IF;

-- dohvati stanje na računu primatelja

SELECT balance INTO receiver\_balance

FROM accounts

WHERE iban = receiver\_iban;

FOR UPDATE;

IF receiver\_balance IS NULL THEN

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Primatelj nije pronađen';

END IF;

-- ako ima dovoljno sredstava pokreni transakciju

IF sender\_balance >= transfer\_amount THEN

-- oduzmi sumu iz računa pošiljatelja

UPDATE accounts

SET balance = balance - transfer\_amount

WHERE iban = sender\_iban;

-- provjera je li ažuriranje računa pošiljatelja prošlo uspješno

SELECT balance INTO new\_sender\_balance

FROM accounts

WHERE iban = sender\_iban;

IF new\_sender\_balance <> sender\_balance - transfer\_amount THEN

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Greška pri ažuriranju računa pošiljatelja';

END IF;

-- dodaj sumu na račun primatelja

UPDATE accounts

SET balance = balance + transfer\_amount

WHERE iban = receiver\_iban;

-- provjera je li ažuriranje računa primatelja prošlo uspješno

SELECT balance INTO new\_receiver\_balance

FROM accounts

WHERE iban = receiver\_iban;

IF new\_receiver\_balance <> receiver\_balance + transfer\_amount THEN

ROLLBACK;

SIGNAL SQLSTATE '45000' SET MESSAGE\_TEXT = 'Greška pri ažuriranju računa primatelja';

END IF;

ELSE

-- ako nema dovoljno sredstava na računu vrati na početno stanje

ROLLBACK;

END IF;

END $$

DELIMITER ;

-- primjer poziva procedure

CALL make\_transaction('DE1234567890', 'DE0987654321', 100.00);

-- primjer funkcije za dohvat stanja računa

DELIMITER $$

CREATE FUNCTION get\_balance(input\_iban VARCHAR(34))

RETURNS DECIMAL(15, 2)

BEGIN

DECLARE balance DECIMAL(15, 2);

-- upit za dohvat stanja računa

SELECT balance INTO balance

FROM accounts

WHERE iban = input\_iban;

-- provjera je li dohvat podataka bio uspješan

IF balance IS NULL THEN

-- ako nije vraćamo null

RETURN NULL;

END IF;

-- povrat podatka

RETURN balance;

END $$

DELIMITER ;

-- primjer poziva funkcije

SELECT get\_balance('HR4890942042048');

**SQL migracije**

* prijenos sheme (database schema migration - tablice, indeksi, veze) i podataka (sql data migration - podaci, retci) iz jedne u drugu (početne u ciljnu) bazu

**Laravel migracije**

* služe za definiranje sheme baze podataka određene aplikacije (kreiranje i modificiranje tablica/entiteta i njenih atributa) sa ciljem olakšavanja prijenosa i rada u timu (primjerice laka dostupnost najnovije verzije baze i u slučaju promjena od strane drugog člana tima koji radi na istoj aplikaciji)

https://laravel.com/docs/11.x/migrations#introduction

// kreiranje nove baze prema shemi u database/migrations folderu

php artisan migrate

// brisanje stare i kreiranje nove baze

php artisan migrate:fresh

// brisanje stare, kreiranje nove baze te punjenje podacima prema shemi u database/seeders i database/factories folderima

php artisan migrate:fresh --seed

**Laravel MVC arhitektura**

Laravel koristi MVC arhitekturu za jasno odvajanje odgovornosti i bolju organizaciju koda, čineći razvoj aplikacija lakšim i održivijim.

Model (M) - odgovoran za pohranu i manipulaciju podacima

View (V) - odgovoran za prikaz podataka korisnicima putem HTML-a

Controller (C) - odgovoran za logiku aplikacije, povezuje modele i prikaze

**Laravel metode za dohvat podataka iz baze**

| **Metoda** | **Opis** | **Vraća** |
| --- | --- | --- |
| all() | dohvaća sve zapise iz tablice | kolekcija svih modela tablice |
| get() | dohvaća kolekciju zapisa s mogućim uvjetima | kolekcija modela temeljenih na upitu nad tablicom |
| first() | dohvaća prvi zapis koji odgovara uvjetima | jedan model ili null |
| find() | dohvaća zapis po njegovom primarnom ključu (ID) | jedan model ili null |

all() - kada želite dohvatiti sve zapise bez ikakvih uvjeta ili filtera

get() - kada trebate dohvatiti kolekciju zapisa s određenim uvjetima (primjerice where)

first() - kada vam treba prvi zapis koji odgovara određenim uvjetima (obično uz where)

find() - kada imate primarni ključ (ID) i želite dohvatiti specifičan zapis (primjer find($id))

**Laravel funkcije za prosljeđivanje podataka iz controllera u view**

view('view\_name', $data) - osnovna metoda za prosljeđivanje podataka

view()->with() - view sa dodatnom metodom za prosljeđivanje podataka

view()->share() - za dijeljenje globalnih podataka sa svim pogledima

compact() - prečica za prosljeđivanje varijabli u pogled

session() - za podatke koji se čuvaju u sesiji između zahtjeva

// podatci koji se šalju u pogled mogu biti bilo kojeg tipa, skalarni (brojke, string, bool), polja (indeksirana, asocijativna), objekti (instance klasa, kolekcije)

**Blade {{ }} sintaksa**

{{ }} u Bladeu koristi se za echo (ispisivanje) podataka/varijabli

// Prikazivanje varijable u Bladeu

<h1>{{ $title }}</h1>

**Laravel Dusk - click()**

Laravel Dusk je alat za testiranje koji omogućava interakciju sa web stranicama imitiranjem stvarnog korisnika.

Njegova funkcija click() koristi se za simuliranje klika na HTML element na stranici prilikom izvođenja automatiziranih testova korisničkog sučelja (UI testova).

**PHP Unit config xml**

* napraviti PHPUnit config xml koji će napraviti include/exclude određenih datoteka

https://docs.phpunit.de/en/10.5/configuration.html#the-exclude-element

https://laraveldaily.com/lesson/testing-laravel/db-configuration-refreshdatabase-phpunit-xml-env-testing

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

<phpunit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:noNamespaceSchemaLocation="./vendor/phpunit/phpunit/phpunit.xsd"

bootstrap="vendor/autoload.php"

colors="true"

convertErrorsToExceptions="true"

convertWarningsToExceptions="true"

convertNoticesToExceptions="true"

stopOnFailure="false">

<!-- definiranje direktorija sa testovima koji će se izvoditi -->

<testsuites>

<testsuite name="Unit">

<directory>tests/Unit</directory>

</testsuite>

<!-- može se dodati više od jedne testne konfiguracije -->

<testsuite name="Feature">

<directory>tests/Feature</directory>

</testsuite>

</testsuites>

<!-- definiranje koji direktoriji će biti uključeni tokom izvođenja testova -->

<source>

<include>

<directory>app</directory>

</include>

<!-- primjer isključivanja direktorija/filea iz provjere -->

<exclude>

<directory suffix=".php">app/Providers</directory>

<file suffix=".php">app/Providers/AppServiceProvider.php</file>

</exclude>

</source>

<!-- za env varijable umjesto definiranja u phpunit.xml fileu možemo koristiti .env.testing file -->

<php>

<env name="APP\_ENV" value="testing"/>

<env name="APP\_MAINTENANCE\_DRIVER" value="file"/>

<env name="BCRYPT\_ROUNDS" value="4"/>

<env name="CACHE\_STORE" value="array"/>

<env name="DB\_CONNECTION" value="mysql"/>

<env name="DB\_DATABASE" value="algebra"/>

<env name="MAIL\_MAILER" value="array"/>

<env name="PULSE\_ENABLED" value="false"/>

<env name="QUEUE\_CONNECTION" value="sync"/>

<env name="SESSION\_DRIVER" value="array"/>

<env name="TELESCOPE\_ENABLED" value="false"/>

</php>

</phpunit>

**Continuous integration**

* CI/CD pipeline (continuous integration/delivery/deployment) uvodi stalnu automatizaciju i kontinuirani nadzor tokom kompletnog životnog ciklusa aplikacije, od faza integracije i testiranja do isporuke i primjene
* CI je praksa prilikom razvoja aplikacije gdje programeri redovito (ponekad svakodnevno) dodaju vlastite promjene koda na zajednički repozitorij, nakon čega se aplikacija gradi te izvode testovi
* obavezni koraci koje bi trebalo dodati u CI (Continuous Integration) pipeline:
  + izvrtiti testove i vidjeti da li prolaze
  + statički analizirati kod te validirati da nema nikakvih pogrešaka
  + napraviti cache konfiguracijskih datoteka projekta te provjeriti da nema pogrešaka

https://group.miletic.net/hr/nastava/materijali/web-kontinuirana-integracija/#tijek-rada-kontinuirane-integracije-12

**Laravel kontroler i auth middleware (bearer token, API JSON response sa status kodom)**

* napravljeno prema ovim primjerima (ovo je u biti jednostavna simulacija Laravel Passport/Sanctum)

<!-- kontroler i rute -->

https://dev.to/thatcoolguy/token-based-authentication-in-laravel-9-using-laravel-sanctum-3b61

<!-- middleware -->

https://stackoverflow.com/questions/58730579/laravel-bearer-token-authentication

https://laravel.com/docs/11.x/middleware#defining-middleware

* Auth kontroler sa register/login/logout metodama (treba ga kreirati sa naredbom "php artisan make:controller AuthController")

<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Auth;

class AuthController extends Controller

{

public function register(Request $request)

{

$validatedData = $request->validate([

'username' => 'required|string|max:255|unique:users',

'email' => 'required|email|max:255|unique:users',

'password' => 'required|Rules\Password::defaults()',

]);

$user = User::create([

'username' => $validatedData['username'],

'email' => $validatedData['email'],

'password' => Hash::make($validatedData['password']),

]);

return response()->json([

'username' => $user->username,

'email' => $user->email,

'token' => $user->createToken('auth\_token')->plainTextToken,

], 201);

}

public function login(Request $request)

{

$validatedData = $request->validate([

'username' => 'required|string|max:255|Rule::unique('users')->ignore($this->route('user'))',

'email' => 'required|email|max:255|Rule::unique('users')->ignore($this->route('user'))',

'password' => 'required|Rules\Password::defaults()',

]);

$user = User::where('username', $validatedData->username)->orWhere('email', $validatedData->email)->first();

if (!$user || !Hash::check($validatedData->password, $user->password)) {

return response()->json([

'message' => ['Username or password incorrect'],

], 401);

}

$user->tokens()->delete();

return response()->json([

'status' => 'success',

'message' => 'User logged in successfully',

'name' => $user->name,

'token' => $user->createToken('auth\_token')->plainTextToken,

], 201);

}

public function logout(Request $request)

{

$request->user()->currentAccessToken()->delete();

return response()->json([

'status' => 'success',

'message' => 'User logged out successfully'

], 201);

}

}

* Middleware koji provjerava korisnikov token (treba ga kreirati sa naredbom "php artisan make:middleware AuthToken")

<?php

use App\Models\User;

use Closure;

use Illuminate\Http\Request;

use Symfony\Component\HttpFoundation\Response;

class AuthToken

{

/\*\*

\* @param \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response) $next

\*/

public function handle(Request $request, Closure $next): Response

{

$token = $request->header('Authorization');

if (User::where('auth\_token', $token)->first()) {

return $next($request);

}

return response()->json([

'message' => 'Unauthenticated'

], 403);

}

}

* te potom registrirati u bootstrap/app.php, novu middleware klasu dodajemo u api grupu i dajemo joj alias (naredbu treba ulančati/nadodati između configure i create metoda klase Application)

->withMiddleware(function (Middleware $middleware) {

$middleware->api(prepend: [AuthToken::class,])

->alias(['auth.token' => AuthToken::class,]);

})

* Rute za metode Auth kontrolera (treba omogućiti vidljivost, odnosno napraviti "publish" api.php filea u routes folderu sa naredbom "php artisan install:api")

<?php

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\AuthController;

Route:controller(AuthController::class)->group(function () {

Route::post('/register', 'register')->name('register');

Route::login('/login', 'login')->name('login');

Route::logout('/logout', 'logout')->name('logout')->middleware('auth.token');

});

**Spajanje domene s poslužiteljem**

1. Kupiti web server (VPS) i domenu od nekog pružatelja usluga (primjerice Hertzner) i registrara za domene (primjerice Cloudflare).

2. Postaviti web server, instalirati aplikacije (primjerice LAMP/LEMP stack na Linux serveru) i provjeriti da li ispravno radi, te je li dostupan na internetu.

Ako se koristi dijeljeno hostiranje, VPS ili cloud hosting, pružatelj hostinga bi trebao imati upute za postavljanje servera.

Javna IP adresa servera dobije se od strane pružatelja hostinga (ali se može saznati i pomoću naredbe 'curl ifconfig.me')

3. Prijaviti se na upravljačku ploču registrara te ažurirati DNS postavke, ovo je ključan korak u povezivanju domene sa serverom gdje se domena usmjerava na IP adresu web servera.

Na DNS management ili name server:

Dodati A zapis koji će usmjeriti domenu na IP adresu servera:

Tip: A

Ime/Host: @ (ovo usmjerava na osnovnu domenu, npr. ime\_domene.com) ili www (ako koristimo poddomenu, npr. www.ime\_domene.com)

Vrijednost/Points to: javna IP adresa servera (npr. 192.0.2.123).

TTL (Time to Live): može se ostaviti zadano, ili postaviti na npr. 3600 sekundi (1 sat).

Opcionalno, može se dodati CNAME zapis za www (ako želimo da www.ime\_domene.com također bude funkcionalno):

Tip: CNAME

Ime/Host: www

Vrijednost/Points to: ime\_domene.com

Primjer DNS zapisa:

A Zapis:

@ → 192.0.2.123

CNAME Zapis (opcionalno):

www → ime\_domene.com

4. Pričekati propagaciju DNS-a, promjene mogu potrajati od nekoliko minuta do nekoliko dana da se potpuno propagiraju kroz internet (obično 1-2 sata)

5. Konfigurirati web server da prepozna domenu (opcije Apache ili Nginx)

Apache:

- uredite Apache konfiguracijsku datoteku (/etc/apache2/sites-available/ime\_domene.conf):

<VirtualHost \*:80>

ServerName ime\_domene.com

ServerAlias www.ime\_domene.com

DocumentRoot /var/www/ime\_domene

ErrorLog ${APACHE\_LOG\_DIR}/error.log

CustomLog ${APACHE\_LOG\_DIR}/access.log combined

</VirtualHost>

- omogućite stranicu: sudo a2ensite ime\_domene.conf

- ponovno učitajte Apache: sudo systemctl reload apache2

Nginx:

- uredite Nginx konfiguracijsku datoteku (/etc/nginx/sites-available/ime\_domene):

server {

listen 80;

server\_name ime\_domene.com www.ime\_domene.com;

root /var/www/ime\_domene;

index index.html;

access\_log /var/log/nginx/ime\_domene\_access.log;

error\_log /var/log/nginx/ime\_domene\_error.log;

}

- kreirajte simboličku poveznicu: sudo ln -s /etc/nginx/sites-available/ime\_domene /etc/nginx/sites-enabled/

- ponovno učitajte Nginx: sudo systemctl reload nginx

6. Testirati domenu nakon propagacije DNS promjena, otvorimo preglednik i pokušamo otvoriti url (primjerice http://ime\_domene.com) kako bi provjerili je li povezana sa serverom i prikazuje li web stranicu

7. Dodatni koraci:

- postavljanje SSL-a (secure sockets layer) kako bi mogli koristiti HTTPS (potrebno je dobaviti SSL certifikat i postaviti ga na server)

- postavljanje mail servera kako bi mogli slati ili primati e-mailove putem domene (potrebno je postaviti MX zapise u DNS-u i konfigurirati mail server)

Ovaj proces će povezati domenu sa web serverom, a web stranica bi trebala biti dostupna prilikom pristupa domeni u web pregledniku.

**Instalacija wsl i ubuntu na virtualnim Windowsima**

* upute za instalaciju wsl i Ubuntu iz command prompta (Windows+R, pa upišite cmd, te potom ctrl+shift+enter da bi ga otvorili sa administratorskim ovlastima)

https://learn.microsoft.com/en-us/windows/wsl/install

* nakon toga trebate spojiti wsl sa VS Code prema sljedećim uputama (koristite "from VS Code" dio)

https://code.visualstudio.com/docs/remote/wsl

* sljedeće pristupite Ubuntu putem terminala na VS Code te napravite naredbu za update i upgrade

sudo apt-get update && sudo apt-get upgrade -y

* zatim kopirajte .setup.sh sa gita (imate link dolje) i pohranite u setup.sh na Ubuntu (sa sudo nano), dajte mu 777 ovlasti (sudo chmod 777), te ga pokrenite kako bi instalirali LAMP stack (php, mysql, apache, composer)

https://github.com/adobrini-algebra/radno\_okruzenje/blob/master/setup.sh

sudo nano setup.sh

sudo chmod 777 setup.sh

setup.sh

**Instaliranje Laravel projekta pomoću Composera**

* s obzirom da već imamo instalirane php i composer (ako smo instalirali LAMP stack pomoću setup.sh datoteke), Laravel možemo instalirati ovom naredbom:

composer global require laravel/installer

* nakon toga kreiramo novu Laravel aplikaciju sa naredbom:

laravel new ime\_aplikacije

* gdje ćete moći odabrati niz opcija, a po instalaciji trebate izmijeniti .env file, itd., detaljnije upute na linku (Laravel dokumentacija):

https://laravel.com/docs/11.x/installation#installing-php

**Postavljanje aplikacije na server**

* ulogirajte se na udaljeni server sa danim podacima (username i password)

username@ip\_adresa\_servera

<!-- nakon čega će vas tražiti password -->

* na serveru najprije treba omogućiti OpenSSH (ako već nije), te dodati vlastiti SSH ključ prilikom spajanja sa desktop (u našem slučaju wsl) Linuxa na simulirani udaljeni Linux server (primjer za to, uključujući i kako stvoriti SSH ključ, te onemogućiti običan password login, možete vidjeti na linkovima dolje)

https://ubuntu.com/server/docs/openssh-server

https://www.digitalocean.com/community/tutorials/initial-server-setup-with-ubuntu

https://www.youtube.com/watch?v=3FKsdbjzBcc

* nakon što ste se spojili sa vlastitog linuxa na udaljeni server, trebate instalirati PHP i Composer (eventualno još i Docker ili samostalno mysql/neku drugi database provider, te apache/nginx), nakon čega ćete imati potpuno spreman "udaljeni" Linux server za pokretanje vaših aplikacija

https://github.com/bozidar09/backend\_developer/blob/master/radno\_okruzenje/setup.sh

sudo apt install -y php libapache2-mod-php php-mysql php-pdo php-intl php-gd php-xml php-json php-mbstring php-tokenizer php-fileinfo php-opcache

php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"

php -r "if (hash\_file('sha384', 'composer-setup.php') === 'dac665fdc30fdd8ec78b38b9800061b4150413ff2e3b6f88543c636f7cd84f6db9189d43a81e5503cda447da73c7e5b6') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP\_EOL;"

sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer

php -r "unlink('composer-setup.php');"

* opcija 1 - prebacivanje aplikacije na udaljeni server naredbom rsync -a
* prije prebacivanja trebate naredbom chown dobiti ovlast nad odredišnim direktorijem na serveru u kojeg želite spremiti aplikaciju (primjer slučaja ako aplikaciju želite prebaciti u odredišni direktorij /var/www/)

sudo chown -R algebra:algebra /var/www/

* nakon toga koristimo naredbu 'rsync -a' kako bi aplikaciju prebacili na udaljeni server (uz napomenu da nam trebaju sudo ovlasti ako na server želimo prebaciti i datoteke kojima vlasnik nije korisnik 'algebra')

sudo rsync -a /var/www/backend\_developer/laravel-videoteka algebra@192.168.1.225:/var/www/

* opcija 2 - prebacivanje aplikacije na udaljeni server pomoću git clone
* prvo trebate kopirati link projekta sa githuba koji želite klonirati, te na serveru napraviti git clone

git clone link\_projekta

* nakon toga, treba prekopirati .env.example file u .env te dodati APP\_KEY

echo $UID

cp .env.example .env

php artisan key:generate

* zatim sa composer install treba dodati vendor folder koji nedostaje (eventualno instalirati i zip, te unzip ako fale)

sudo apt-get install zip && sudo apt-get install unzip

composer install

* pokretanje aplikacije na udaljenom serveru
* kako bi spriječili greške sa file permissionima, trebate odraditi ove naredbe:

sudo chown -R algebra:www-data storage/ bootstrap/cache/

sudo chmod -R 775 storage/ bootstrap/cache/

* ako u projektu koristite custom javaScrip (Tailwind i slično) onda trebate odraditi npm naredbe

npm install

npm run build

* zatim napraviti provjeru i po potrebi izmjene .env filea (user\_id, ime baze, username, password i slično)

echo $UID

sudo nano .env

* naposlijetku trebate napraviti symlink, te napuniti bazu sa podacima

php artisan storage:link

php artisan migrate --seed

* opcija - pokretanje aplikacije pomoću dockera na udaljenom serveru
* najprije trebate instalirati docker na Linux udaljenog servera (upute na linku, opcionalno možete i dodati docker u sudo grupu)

https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-22-04

* zatim trebate provjeriti portove u .env fileu (app\_port, forward\_db port i slično) i eventualno ih zamijeniti (najjednostavnije +1) ako želite upogoniti više docker aplikacija (jer svaka treba raditi na vlastitom portu)
* sljedeće trebate pokrenuti docker, te instalirati potrebne containerse (sa docker ps naredbom možete provjeriti koji containeri trenutno rade na serveru)

docker compose up -d

docker ps

* naposlijetku trebate ući u app docker container te napraviti symlink i pokrenuti migraciju

docker compose exec -it app bash

php artisan storage:link

php artisan migrate --seed

* nakon ovoga bi primjerice aplikaciji sa ip adresom 'udaljenog servera' 192.168.1.225 na portu 8000 trebali moći pristupiti u web browseru sa:

192.168.1.225:8000