



# Architektúra SSR Laravel – úvod

kurz Základy  
webových technológií

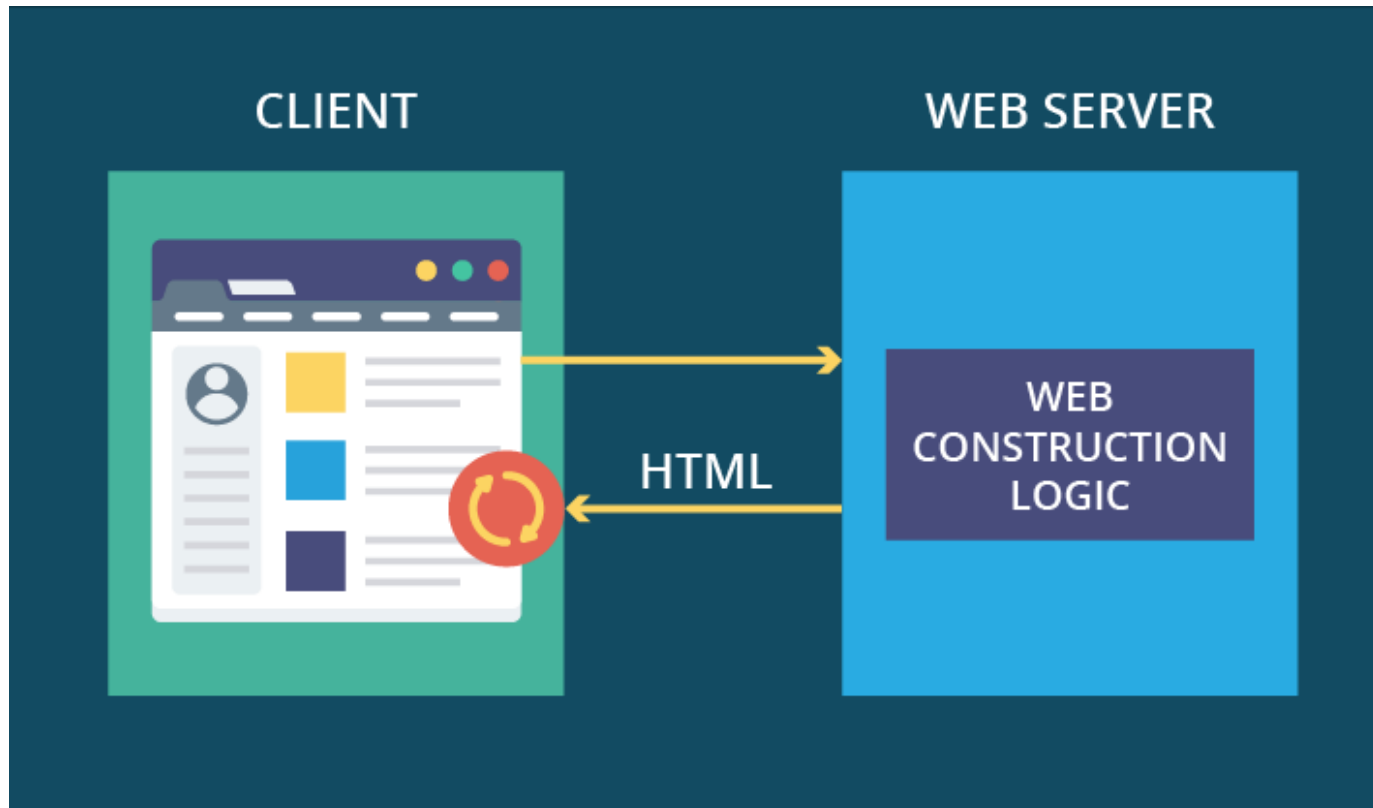
Eduard Kuric

# Architektúry web aplikácií

- **tri základné architektúry web aplikácií**
- **zostavenie/generovanie stránok na strane servera**
  - angl. Server-Side Rendering (server-side HTML)
- dynamicky načítavané vybrané fragmenty/oblasti stránok cez async JavaScript
  - angl. JS widgets
- **zostavenie/generovanie stránok na strane klienta**
  - angl. client-side rendering, tiež Single Page Application

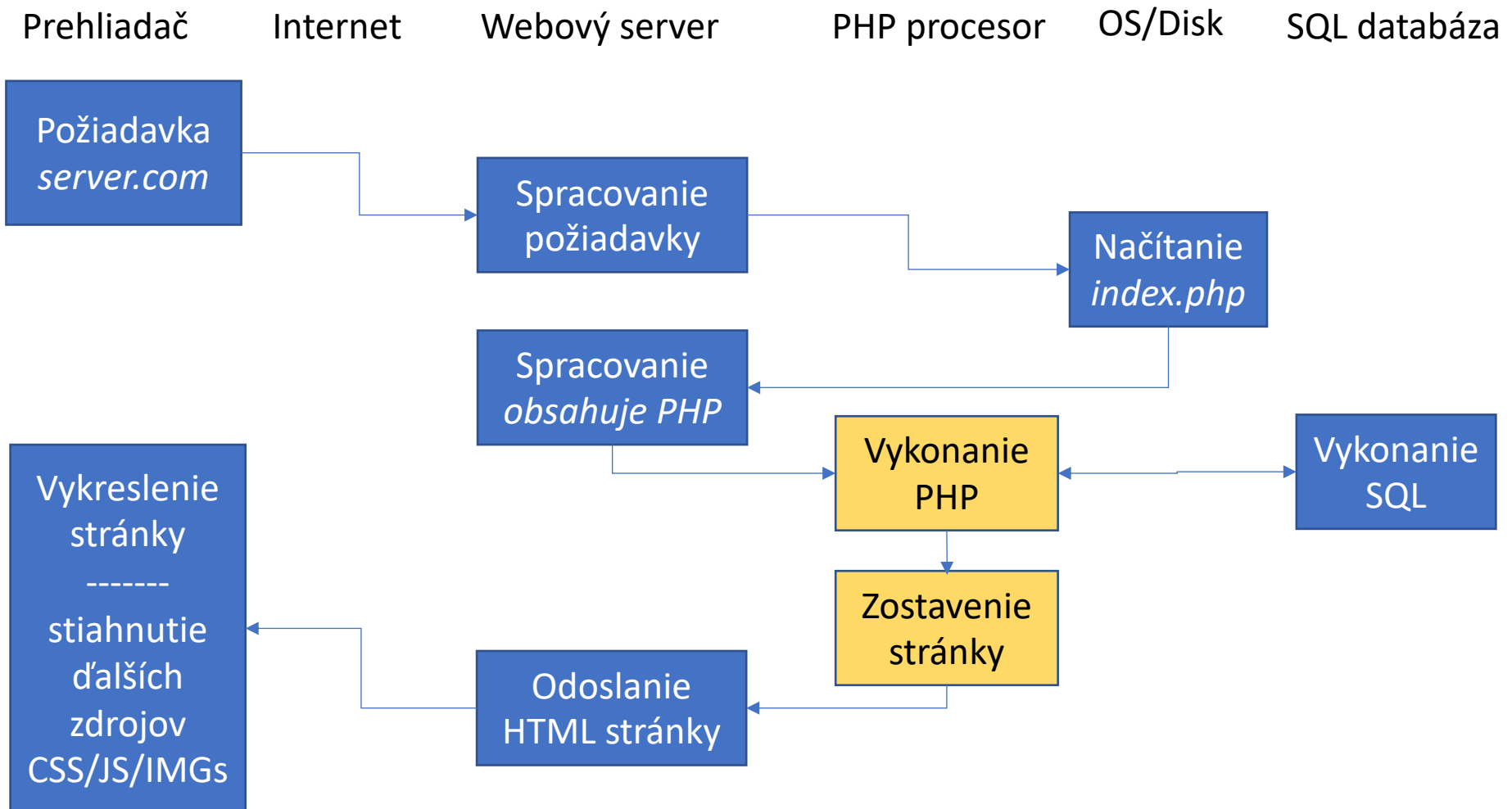
# Server-side rendering

- **server generuje HTML stránky a posíla klientovi**



# Dynamické stránky

klient/server požiadavka/odpoveď



# HTTP /RFC 2616

- **Hypertext Transfer Protocol**
- **internetový protokol na výmenu hypertextových dokumentov (HTML)**
- rozšírenie MIME (**M**ultipurpose **I**nternet **M**ail **E**xtensions) umožňuje prenášať akýkoľvek súbor
- implicitný port 80, https 443
- **používa URL** (Uniform Resource Locator) - jednoznačné umiestnenie zdroja

# Zostavenie na strane servera

- najstaršia architektúra, zostavenie prebieha na výkonných serveroch
- na strane servera je veľká variabilita technológií/jazykov
- **plus:** bezpečnosť - biznis logika aplikácie je na serveri, vhodné aj z pohľadu „ochrany“ kódu pred neoprávneným použitím 3. stranou
- **mínus:** medzi klientom a serverom sa posiela množstvo dát, (ktoré sme už predtým prijali)

# Vystačíme si s JavaScriptom?

- biznis logika bola a je bežne napísaná v
  - [PHP](#)
  - Ruby
  - Python
  - Perl
- dnes už môže byť napísaná v JavaScripte
  - [Node.js](#), nadstavby (rámce) [Express](#), [Meteor](#), [Adonis](#), [Sails](#)

# PHP generuje 80% Webu

- najpopulárnejší, alebo skôr najrozšírenejší jazyk na strane servera
- Facebook, Wikipedia, Flickr, Wordpress, Aliexpress sú v PHP



# Laravel (11)

- [najpopulárnejší PHP rámec](#)
- významne **zjednodušuje vývoj**, netreba rutinné veci znovu programovať
  - autentifikáciu
  - prácu s DB cez objektovo-relačné mapovanie
  - caching
  - ...
- MVC architektúra: Model-View-Controller
  - oddelený dátový model, rozhranie, a riadiaca/biznis logika

# Základné požiadavky

- PHP >= 8.2
- CType PHP Extension
- cURL PHP Extension
- DOM PHP Extension
- Fileinfo PHP Extension
- Filter PHP Extension
- Hash PHP Extension
- Mbstring PHP Extension
- OpenSSL PHP Extension
- PCRE PHP Extension
- ...

<https://laravel.com/docs/11.x/deployment>

# Webový server

- [Windows: WAMP server](#)
  - Apache server
  - PHP
  - MySQL DB (phpMyAdmin)
- Linux: LAMP
  - <https://howtoubuntu.org/how-to-install-lamp-on-ubuntu>
- Mac OS: brew + phpbrew
  - <https://phpbrew.github.io/phpbrew/>

**Laravel built-in server:** `php artisan serve`

# Composer – inštalácia

- je **nástroj na správu knižníc** a iných závislostí (dependency manager) pre PHP
- znovupoužitie existujúcich riešení
- jednoduchá aktualizácia použitých závislostí
- <https://getcomposer.org/>
- <https://getcomposer.org/download/>
- <https://getcomposer.org/doc/>

# Composer – príklad závislosti

- napr. potreba pársovať XML dokument
  - nie je potrebné opätovne vymýšľať koleso
  - <https://github.com/orchestral/parser>
  - v projekte spustíte
    - `composer require "orchestra/parser=~3.0"`
    - composer stiahne knižnicu, doplní konf. súbor o závislosť
    - aparát knižnice je v projekte plne k dispozícii
- súčasťou projektu sú composer súbory:
  - `composer.json` - obsahuje použité závislosti
  - `composer.lock` – obsahuje verzie použitých závislostí, ktoré majú byť v projekte použité
- na stránkach knižníc je spravidla postup inštalácie cez composer

# Inštalácia Laravel aplikácie

- **Laravel nainštalujeme cez composer**
  - spustite CLI
  - nastavte sa do priečinka, v ktorom bude priečinok aplikácie (napr. koreňový priečinok servera)  
*priečinok aplikácie nevytvárajte, vytvorí composer*

```
composer create-project laravel/laravel  
myapp
```

- myapp – názov priečinka, v ktorom composer vytvorí základnú štruktúru a súbory pre Laravel aplikáciu
- ```
composer create-project  
laravel/laravel:^10.0 example-app2
```

# Spustenie zabudovaného servera

- na spustenie aplikácie je možné použiť zabudovaný webový server
  - cez CLI sa nastavte sa do priečinka aplikácie
  - spustite `php artisan serve`
- výstup príkazu by mal vyzeráť podobne:  
Laravel development server started:  
<http://127.0.0.1:8000>
- po zadaní URL v prehliadači:  
<http://127.0.0.1:8000>
  - by sa mala zobrazíť úvodná stránka základnej Laravel aplikácie

# Štruktúra aplikácie

- `app` - hlavný kód aplikácie, aplikačná logika
- `bootstrap` – obsahuje `app.php`, ktorý zavádza (angl. bootstraps) aplikačný rámec (nemá nič s [týmto bootstrapom](#))
- `config` – konfiguračné súbory
- `public` – obrázky, css, javascripty
- `resources` – views - prezentačná vrstva
- `routes` – nastavenie smerovania požiadaviek
- `storage` – kompilované šablóny, cookies, sessions
- `tests` – jednotkové testy
- `vendor` – kód laravelu + závislostí



# Routing - smerovanie

- aparát, ktorý **zabezpečuje mapovanie požiadaviek** (requests) na **konkrétnu aplikačnú logiku** (controller)
- definujú sa v `/routes`

# Controller – aplikačná logika

- na základe požiadavky vykonáva špecifickú aplikačnú/biznis logiku
  - komunikuje s modelmi (databázou)
  - spracováva/transformuje údaje, realizuje výpočty
  - generuje rozhranie (HTML)
  - výsledok vracia prehliadaču
- sú uložené v `/app/Http/Controllers/`

# View – rozhranie

- oddeluje prezentačnú logiku od aplikačnej
- môže obsahovať HTML celej stránky, alebo nejakej jej časti
- medzi HTML sú PHP bloky, v ktorých controller dopĺňa údaje
- sú uložené v `resources/views`

# Prvý controller

- `php artisan make:controller UserController --resource`
  - `app/Http/Controllers/UserController.php`

- `do routes/web.php pridáme:`

```
use App\Http\Controllers\UserController;  
Route::resource('users', UserController::class);
```

# Routing – Controller – View

- ručne vytvoríme view

resources/views/users/**index.blade.php**

- vložíme obsah:

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
    <meta charset="utf-8">
```

```
    <title>Users Index</title>
```

```
</head>
```

```
<body><h1>Users Index</h1></body>
```

```
</html>
```

# Routing – Controller – View /3

- do UserController v metóde index vložíme:

```
return view('users.index');
```

- cez CLI spustíme

```
php artisan serve
```

- v prehliadači zadáme

```
http://127.0.0.1:8000/users
```

# Prvý controller /2 CRUD

- `Route::resource('users', 'UserController');`

<i>HTTP metóda</i>	<i>URL</i>	<i>Akcia</i>
• GET	/users	index
• GET	/users/create	create
• POST	/users	store
• GET	/users/{user}	show
• GET	/users/{user}/edit	edit
• PUT/PATCH	/users/{user}	update
• DELETE	/users/{user}	destroy

# REST architektúra (API)

- Representational State Transfer
  - architektúra, umožňuje prístupovať k údajom na určitom mieste (endpoint) pomocou štandardných metód HTTP
  - jednotný a jednoduchý prístup k zdrojom
    - dáta, stavy aplikácie
  - všetky zdroje majú vlastný identifikátor URI
  - definuje 4 základné metódy na prístup ku zdrojom, známe ako CRUD
    - Create – vytvorenie údajov POST (HTTP)
    - Retrieve – získanie údajov GET (HTTP)
    - Update – zmena údajov PUT (HTTP)
    - Delete – vymazanie údajov DELETE (HTTP)

<https://developer.twitter.com/en/docs/accounts-and-users/create-manage-lists/overview>



# Retrieve - GET požiadavka HTTP

- **na získanie dát** (zdroja - dokumentu...)
- používa URL, odovzdáva parametre
- má limitovanú dĺžku
- môže byť cachovaná, uložená v histórii prehliadača
- **nikdy by nemala byť použitá na prenos citlivých údajov**

# Create - POST požiadavka HTTP

- **na posielanie/vytváranie dát**
- „nemá“ obmedzenie na dĺžku dát
- dáta nie sú cachované, nezostávajú v histórii prehliadača
- parametre z URL sa odosielajú v tele správy

# Delete - DELETE požiadavka HTTP

- volanie je podobné ako pri metóde GET
  - v praxi je problematické vyvolať metódu DELETE
    - HTML formuláre sú obmedzené iba na metódy GET a POST
    - alt.: metóda POST so skrytým parametrom, ktorý indikuje, že ide o metódu DELETE
- ```
<input name="_method" type="hidden" value="DELETE">
```

# Update – PUT požiadavka HTTP

- volanie je podobné ako pri metóde POST,  
**ale voláme konkrétnu URI - konkrétneho zdroja,  
ktorý chceme zmeniť**
- v praxi je problematické vyvolať metódu PUT
  - HTML formuláre sú obmedzené iba na metódy GET a POST
  - alt.: metóda POST so skrytým parametrom, ktorý indikuje, že ide o metódu PUT

```
<input name="_method" type="hidden" value="PUT">
```

# REST zhrnutie

- spolu s formátom JSON je štandardom pre API webových služieb
  - významne sa nelíši od štandardného volania a získavania dát prostredníctvom HTTP protokolu - zovšeobecňuje
  - umožňuje CRUD operácie prostredníctvom štandardných HTTP požiadaviek
- moderné rámce SSR pomáhajú vytvárať REST API
- bezstavové

# Profil konkrétného používateľa

`http://127.0.0.1:/8000/users/2`

- v `UserController` použijeme metódu:

```
public function show($id)
{
    return view('users.profile', ['id' => $id]);
}
```

# Profil konkrétného používateľa /2

- **vytvoríme view** `users/profile.blade.php`:

```
<!DOCTYPE html>
```

```
<html lang="en">
```

```
<head>
```

```
    <meta charset="utf-8">
```

```
    <title>User profile</title>
```

```
</head>
```

```
<body><h1>Hello, user {{ $id }}</h1></body>
```

```
</html>
```

# Model

- tabuľka v datábaze má zodpovedajúci model
  - triedu v Laravel projekte
- model = riadok/inštancia tabuľky
- sú uložené v `app/`



# Pripojenie na databázu

- **v** / .env súbore nastavme pripojenie na našu DB

DB\_CONNECTION=pgsql

DB\_HOST=127.0.0.1

DB\_PORT=5432

DB\_DATABASE=wtech

DB\_USERNAME=postgres

DB\_PASSWORD=123456

# Pripojenie na databázu

- v `config/database.php` nastavíme pripojenie na databázu pre `pgsql`:

```
'database' => env('DB_DATABASE', 'wtech'),  
'username' => env('DB_USERNAME', 'ekoku'),  
'password' => env('DB_PASSWORD', '123456'),
```

- nezabudnime v `php.ini` odkomentovať `extension`
- `extension=php_pgsql.dll`
- `extension=php_pdo_pgsql.dll`

# Vytvorenie tabuľky users

```
CREATE TABLE "users"  
(  
    id bigserial NOT NULL,  
    name char varying(512) NOT NULL,  
    email text NOT NULL UNIQUE,  
    password char varying(128) NOT NULL,  
    CONSTRAINT users_pkey PRIMARY KEY (id)  
)
```

# Vloženie záznamov do tabuľky

```
INSERT INTO users (name, email,  
password) values ('Eduard Kuric',  
'eduard.kuric@stuba.sk',  
md5('123456'));
```

```
INSERT INTO users (name, email,  
password) values ('Róbert Mak',  
'robert.mak@stuba.sk',  
md5('123456'));
```

# Vytvorenie modelu User

- v app už existuje súbor `User.php`
  - ten zatiaľ premenujeme na `-User.php`

- cez CLI:

```
php artisan make:model User
```

- v app sa vytvorí nový súbor `User.php`

# Zmeny v UserController

- **pod namespace pridáme**

```
use App\Models\User;
```

- **obsah metódy index pozmeníme na:**

```
$usersList = User::all();  
return view('users.index')->with('usersList',  
$usersList);
```

# Zmeny vo view users/index

```
<table>
  <tr><th>ID</th><th>Meno</th><th>Email</th></tr>
  @foreach($usersList as $user)
    <tr>
      <td>{{ $user->id }}</td>
      <td>{{ $user->name }}</td>
      <td>{{ $user->email }}</td>
    </tr>
  @endforeach
</table>
```

# Zmeny v UserController /2

- **obsah metódy show** pozmeníme na:

```
$user = User::find($id);  
return view('users.profile')->with('user',  
$user);
```



# Zmeny vo view users/profile

```
<h1>Hello, {{ $user->name }}</h1>
```

```
<p>Your email is: {{ $user->email }}</p>
```

# Vytvorenie prihlasovania jednoducho

- zmažeme v DB nami vytvorenú tabuľku users
- Nainštalujeme [Starter Kit](#)  
`composer require laravel/breeze --dev`
- Nainštalujeme Node.js: <https://nodejs.org/en/>
- Následne:
- `php artisan breeze:install`
- `npm install`
- `php artisan migrate`
- `npm run dev`

- `php artisan serve`

- vyskúšajme

<http://127.0.0.1:8000/register>

<http://127.0.0.1:8000/login>

# Laravel naming conventions

- názov controllera - jednotné číslo
  - `UserController`, nie `UsersController`
- route - množné číslo
  - `users/26`
- view – množné číslo
  - `users.index`; `users.profile`
- názov tabuľky v DB – množné číslo
  - `users`
- názov zodpovedajúceho modelu – jednotné číslo
  - `User`
- <https://github.com/alexeymezenin/laravel-best-practices#follow-laravel-naming-conventions>

# Stavové kódy HTTP

- súčasť hlavičky odpovede zo serveru na požiadavku
- upresňuje, ako bola odpoveď spracovaná
- **1xx** Informational responses (informačné)
- **2xx** Success (úspech)
- **3xx** Redirect (presmerovanie)
- **4xx** Client error (404 not found)
- **5xx** Server error (500 Internal Server Error)