Dokumentace semestrální práce

KIV/WEB

Zadání 1

Václav Honzík, **A19B0674P**
honzikv@students.zcu.cz, 30. 12. 2020

Použité technologie

Backend běží na Apache HTTP serveru, který obstarává požadavky připojených klientů. Server je ovládán pomocí PHP skriptů, které řídí celý chod aplikace. PHP zde slouží jednak k implementaci samotné logice serveru, tak i k řízení zápisů a čtení z databáze a předávání views uživateli. Views jsou vytvořené pomocí šablonovacího frameworku Twig, který značně usnadňuje generování HTML oproti PHP. Pro ukládání uživatelských dat byla použitá databáze MySQL.

Frontend využívá základní technologie jako HTML, CSS a JS. Pro lepší vzhled jsou využité knihovny Bootstrap a Font Awesome, které obsahují základní nástroje pro tvorbu responsivních stránek. Aby stránky nebyly tolik „statické“ je dále použito JQuery a AJAX (XMLHttpRequest), které v některých částech nahrazuje pouze část stránky, aby se nemusela renderovat celá. JQuery (i AJAX) jsou importované pomocí CDN (content delivery network), protože jsou značně využívané na většině webech, takže by je měl mít uživatel zacachované a tím pádem by měl být web rychlejší (teoreticky).

Pro zálohu a verzování byl použit Github a pro celé sestavení projektu byl použit Composer.

Celkový seznam technologií:

- **Apache HTTP Server** (backend) – HTTP server umožňující celý chod aplikace – zprostředkovává komunikaci s prohlížečem uživatele, uchovává část uživatelských dat
- **MySQL** (backend) – relační databáze pro správu uživatelských dat – uživatelé, jejich příspěvky atd.
- **PHP** (backend) – jazyk pro programování Apache Serveru, komunikace s databází, samotná logika aplikace
- **Twig** (frontend) – šablonovací systém, pomocí kterého lze snadno vygenerovat HTML stránku
- **Bootstrap** (frontend) – jedná se o framework s velkým množstvím užitečných funkcí a komponent pro tvorbu responsivních webů
- **Font Awesome** (frontend) – obsahuje velké množství ikon pro snadnou integraci do aplikace
- **HTML** + **CSS** + **JS** (frontend) – tvoří view, soubory, které získá uživatel a jediné soubory, kterým rozumí internetový prohlížeč
- **JQuery** (frontend) – JS knihovna pro zjednodušení práce s DOM elementy v HTML dokumentu
- **AJAX / XMLHttpRequest** (frontend) – Ajax usnadňuje komunikaci s backendem, protože pomocí něj lze asynchronně odesílat požadavky na server a lze tak dynamicky aktualizovat pouze část stránky
- **Composer** – slouží pro získání závislostí (twig) a obsahuje navíc autoloader pro snazší import tříd pomocí namespaces
- **Github** – pro verzování aplikace - https://github.com/honzikv/codeguideswebapp

Struktura aplikace

Komponenty aplikace jsou rozdělené do složek podle logických celků:

- **config** – složka s konfigurací serveru – obsahuje údaje pro přístup do databáze
- **controller** – obsahuje Controller objekty pro zpracování požadavků uživatele
- **core** – obsahuje „jádro“ aplikace – základní třídy pro chod serveru – přístup k databázi, wrappery pro request, post, session objekty, router pro přesměrování podle URI apod.
- **db** – obsahuje skripty pro naplnění databáze
- **model** – model obsahuje jednotlivé modely v aplikaci
- **public** – jediná složka, ze které (kromě vygenerovaného HTML) může klient získat data – obsahuje index.php pro obsloužení požadavku, a potřebné soubory jako CSS, JS a bootstrap
- **userdata** – obsahují uživatelská data – soubory článků, které lze stahnout
- **(vendor)** – složka, která se vygeneruje composerem
- **view** – obsahuje jednotlivé views v aplikaci

Architektura aplikace

Aplikace používá architekturu **MVC**, která je standardní architekturou při tvorbě webových aplikací. Tato architektura umožňuje oddělení jednotlivých vrstev tak, abychom získali přehledný a udržitelný kód. Architektura se skládá ze tří základních komponent – **model**, **view** a **controller**.

Model má na starost business logiku aplikace a reprezentaci dat pro daný problém. V případě této práce slouží model ke komunikaci s databází a k validaci dat. Pro každou uživatelskou interakci existuje jedna třída, která v sobě uchovává data od uživatele a provádí validaci. Samozřejmě pro větší aplikaci by bylo nutné model rozdělit do menších celků – např. DTO (Data Transfer Object – pouze přenáší data v aplikaci) a DAO (Data Access Object – pouze obstarává data – např. z databáze a mapuje je na DTO).

View reprezentuje samotné uživatelské rozhraní – v případě webové aplikace se zpravidla jedná o HTML, CSS a JS soubory. View by měla být jediná část, ke které má uživatel přístup. Zde jsou dané views v aplikaci navíc generovány pomocí šablonového frameworku Twig, který před odesláním view uživateli vytvoří požadované HTML.

Controller obstarává komunikaci s klientem (webovým prohlížečem) – při požadavku vrátí danou odpověď. Často je daným požadavkem získat view, ke kterému controller použije model pro získání dat.

Pro zprovoznění celé MVC architektury jsou potřeba další komponenty, které všechny tři prvky spojí, aby mohli komunikovat mezi sebou. Takové komponenty (třídy) jsou umístěné ve složce **core**. **Application** obsahuje funkcionalitu pro přístup k databázi, odesílání odpovědí, příjmání požadavků apod. Tato třída lze získat globálně, čímž se můžeme vyhnout složitému získávání

závislostí pro MVC objekty (např. view by vždy potřeboval nějakým způsobem získat přístup k db, který by se musel předávat konstruktorem).

Třídy **BaseController** a **BaseModel** tvoří základ všech ostatních controllerů a modelů. Obsahují základní funkcionalitu, kterou využívá většina z nich oddědených tříd pro minimalizaci duplikovaného kódu. Dalšími důležitými třídami jsou **Request** a **Response**, které slouží jako wrapper pro superglobální proměnnou $_SERVER. Tyto třídy obsahují užitečné funkce pro získání dat z požadavku a nastavení odpovědi. Další třídou je **Session**, která naopak zabaluje operace nad superglobální proměnnou $_SESSION. Pomocí tohoto objektu můžeme zjistit zda-li je uživatel přihlášený, jeho id, roli apod.

Poslední třídou ze složky core je **Router**. Router slouží ke směrování v aplikaci. Tento objekt je zodpovědný za zavolání správné funkce v controlleru a předání dat. Třída má v sobě uložené asociativní pole, které obsahuje metodu, její URI cestu a callback funkci (funkci controlleru). Tímto způsobem můžeme obsloužit jakýkoliv požadavek, který uživatel zašle. V praxi tak např. pro metodu GET a cestu „/“ router ví, že má zavolat funkci render() ve třídě MainPageController. Samozřejmě pro špatnou cestu nebo metodu se automaticky přesměrujeme na 404.

Jak již bylo zmíněno, uživatel by neměl mít ideálně přístup k žádným souborům, které nejsou view. Toto lze ošetřit buď tím, že na Apache zakážeme pomocí pravidel odesílání souborů s určitým názvem, nebo server spustíme v jiné složce, než je kořen aplikace. Rozhodl jsem se implementovat druhé řešení – tzn. server spustíme ze složky **public** a uživatel by tak měl mít vždy přístup pouze k souborům, které jsou ve složce public.

Závěr

Aplikace by měla splňovat zadání semestrální práce. Backend webu je realizovaný pomocí Apache HTTP serveru, PHP a MySQL. Stránky by měly být odolné vůči základním útokům jako XSS a SQL Injection, nicméně pro reálné použití by potřebovali značně lepší zabezpečení. Web by šel velmi snadno shodit např. pomocí DOS nebo DDOS útoku. Pro frontend jsem použil základní technologie jako HTML, CSS a JS a dále ještě knihovny Twig, jQuery (a AJAX), Bootstrap a Font Awesome, které práci značně usnadňují.