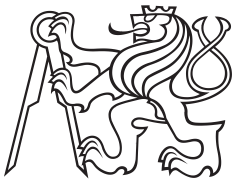
Bakalářská práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra kybernetiky

C++ framework pro vývoj webových aplikací

Adam Novák

Vedoucí: RNDr. Ingrid Nagyová, Ph.D.
Obor: Základy umělé inteligence a počítačových věd
Studijní program: Otevřená informatika
Květen 2022

Poděkování

Děkuji RNDr. Ingrid Nagyové, Ph.D. za vedení mé bakalářské práce, zároveň tak za poskytnuté konzultace a podnětné návrhy, které práci obohatily. Taktéž děkuji rodině a přítelkyni za oporu, kterou mi v době vysokoškolského studia byly.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 20. května 2022

Abstrakt

Tato práce se zabývala návrhem a implementací frameworku pro vývoj webových aplikací v systémovém jazyce C++. Cílem bylo dosáhnout vysoké rychlosti a efektivní práce s pamětí při zpracovávání uživatelských dotazů. V neposlední řadě byl kladen velmi silný důraz na bezpečnost celého řešení. Za tímto účelem byly hlavní části frameworku objektově orientované, řádně zdokumentované, otestované a nabízí kontrolní výpisy o svém stavu do konzole prohlížeče. V závěru práce byla implementace porovnána s nejvíce rozšířenými webovými frameworky a to především v oblasti rychlosti.

Klíčová slova: webové aplikace, C++, framework

Vedoucí: RNDr. Ingrid Nagyová, Ph.D.
České vysoké učení technické v Praze,
Fakulta elektrotechnická,
Katedra počítačů,
Karlovo náměstí 13,
121 35 Praha 2

Abstract

This thesis was concerned with the design and implementation of the framework for development of web applications in the C++ programming language. The aim was to reach a high speed and an effective proceeding with the memory in the course of dealing with the user input. Also, a considerable importance on the safety aspects of the solution was exercised. For this purpose, the main parts of the framework were be object-oriented, properly documented and tested. The main parts of the framework offer a control output on its state to the browser console. In the end of the thesis, the implementation was compared with the mostly used web frameworks and, in particular, the comparison was be made in the line of speed considerations.

Keywords: web applications , C++, framework

Title translation: C++ Framework for Web Application Development

Obsah

1 Úvod **1**

1.1 Kontext problematiky 1

1.2 Cíle projektu 2

2 Základní pojmy **3**

2.1 Softwarový framework 3

2.2 Webový framework 3

2.3 Obecné členění 4

2.4 CGI 4

2.5 Kybernetická bezpečnost 4

3 Analýza problematiky **5**

3.1 Vymezení analýzy 5

3.2 Metody analýzy 6

3.3 Cílová skupina 6

3.4 Licence 6

3.5 Životní cyklus dotazu 7

3.5.1 Laravel 7

3.5.2 Django 8

3.5.3 Nette 8

3.6 HTTP požadavek 9

3.7 URL adresa 11

3.8 Směrování 13

3.9 Návrhové vzory 15

3.10 Databáze 17

3.11 HTTP odpověď 18

3.12 Shrnutí 20

4 Návrh řešení **23**

4.1 Programovací jazyk 23

4.2 Cílová skupina 24

4.3 Licence 24

4.4 Životní cyklus 24

4.5 Architektura 25

4.6 Technický návrh 27

4.6.1 Log 28

4.6.2 Konfigurace 28

4.6.3 HTTP Požadavek 29

4.6.4 Klient 30

4.6.5 URL 30

4.6.6 Směrovač 31

4.6.7 HTTP odpověď 31

4.6.8 Databáze 32

4.6.9 Model-View-Controller 32

5 Implementace **35**

5.1 Struktura 35

5.2 Dokumentace 37

5.3 Průběh implementace 37

6 Ověření **39**

6.1 Použití 39

6.2 Rychlost 41

6.3 Bezpečnost 42

6.4 Rozsah 42

7 Závěr **43**

A Literatura **45**

B Technická dokumentace **49**

B.1 Závislosti 49

B.2 Kompilace 49

B.3 Tvorba komponent 49

C Externí přílohy **53**

C.1 Webový framework 53

C.2 Ukázka použití 53

D Zadání práce **55**

Obrázky

3.1 Výchozí indexový soubor frameworku Laravel 7

3.2 Životní cyklus dotazu mezi klientem a Django 8

3.3 Výchozí indexový soubor frameworku Nette 9

3.4 Atributy a základní přístupné metody analyzovaných request tříd 10

3.5 URI syntax dle RFC3986 [1] . . . 11

3.6 Základní přístupné metody UrlScript třídy v Nette [1] 12

3.7 Schéma umístění směrování uvnitř webového frameworku 13

3.8 Ukázka Továrny objektu RouteList v Nette 13

3.9 Ukázka struktury souboru web.php v Laravel 14

3.10 Ukázka modulu obsahující URL vzory v Django 14

3.11 Základní přístupné metody routovacích tříd Nette a Laravel [1, 2] 15

3.12 Diagram MVC architektury . . . 16

3.13 Diagram MVP architektury . . . 16

3.14 Diagram MTV architektury . . . 16

3.15 Ukázka jazyka SQL 17

3.16 Ukázka databázového SQL dotazu ve frameworku Laravel 17

3.17 Ukázka databázového SQL dotazu ve frameworku Nette 18

3.18 Ukázka hlavičky HTTP odpovědi 19

3.19 Ukázka práce s objektem pro HTTP odpověď v Nette 19

3.20 HTTP odpověď za užití podtřídy v Django 20

3.21 HTTP odpověď bez užití podtřídy v Django 20

3.22 HTTP odpověď ve frameworku Laravel 20

3.23 Zjednodušený životní cyklus analyzovaných webových frameworků 21

4.1 Návrh životního cyklu 25

4.2 MVC architektura obsahující front controller 26

4.3 Ukázka webové aplikace za užití protokolu FastCGI 26

4.4 Technický návrh aplikace 27

4.5 Diagram Log třídy 28

4.6 Ukázka konfiguračního souboru . 28

4.7 Diagram Config třídy 29

4.8 Diagram HTTP Request třídy . . 29

4.9 Diagram Client třídy 30

4.10 Diagram URL třídy 31

4.11 Diagram Router třídy 31

4.12 Diagram HTTP Response třídy 32

4.13 Diagram MVC Controller třídy 32

4.14 Diagram MVC Model třídy . . . 33

4.15 Diagram MVC View třídy 33

5.1 Ukázka jmenných prostorů 35

5.2 Struktura projektu 36

5.3 Ukázka dokumentace funkce napsané v Javadocu 37

6.1 Ukázka stránky Pivovaru BORN 40

B.1 Ukázka uspořádání komponent uvnitř adresáře 50

B.2 Ukázka výčtu ID kontrolérů . . . 50

B.3 Ukázka továrny na směrovač . . . 51
B.4 Ukázka továrny na kontroléry . . 52
B.5 Ukázka výčtu komponent uvnitř souboru CMakeLists.txt 52

Tabulky

3.1 Přehled licencí analyzovaných webových frameworků 6

6.1 Výsledky měření rychlosti webových frameworků 41

Kapitola 1

Úvod

Velké množství aplikací, které byly historicky dostupné pouze v desktopové podobě, se přesouvá do prostředí internetu. K této změně došlo na počátku 21. století s rozvojem komunikačních technologií a jejich začleněním do každodenního života. Z hlediska vývoje se jedná o logický krok, jelikož není nutné vytvářet varianty zdrojového kódu v různých programovacích jazycích pro různé operační systémy. Unifikací a zjednodušením celého vývoje lze zvýšit nejen kvalitu, ale i rychlost a bezpečnost výsledného produktu, což jsou velmi důležité parametry, které jsou často z důvodu nedostatku času zanedbávány. Zároveň je pro uživatele často komfortnější, pokud není nucen projít instalačním procesem, ale může se pohodlně přihlásit skrze webový formulář do svého účtu.

1.1 Kontext problematiky

Kontext problematiky je poměrně rozsáhlý a sahá do několika odvětví. V prostředí internetu jsou historicky ustálené některé technologie, které jsou široce podporované ze strany hostingů a vývojářů, avšak nejsou natolik efektivní, aby mohly být základem aplikací s velkou uživatelskou základnou. Takovým příkladem je například populární jazyk PHP, který je z pohledu vývojáře komfortní, jelikož je interpretovaný a netypový, avšak v některých ohledech velmi neefektivní, a tudíž pro některé projekty nevhodný. Důležitým faktorem se tedy stává zvolený programovací jazyk. Jazyky, které jsou kompilované bývají přirozeně rychlejší a z hlediska ochrany know-how, obsaženého ve zdrojovém kódu, bezpečnější než ty, které jsou interpretované.

Vývojáři zpravidla používají frameworky, aby nemuseli řešit elementární problémy, které jsou s tvorbou aplikací spojeny. Navíc je jejich použitím možné eliminovat nutnost dalších, jinak nezbytných, znalostí, mezi které patří například SQL jazyk, různé protokoly apod. Avšak v případě využití populárních řešení, která jsou často komplexní a prošla si dlouhodobým vývojem ze strany komunity, může dojít k částečnému úpadku rychlosti, neefektivitě a v některých případech i k bezpečnostní chybě.

1.2 Cíle projektu

Hlavním cílem projektu je navrhnout a implementovat takový server-side webový framework, který se bude orientovat především na poskytnutí maximálního výkonu za minimálního využití systémových zdrojů a na bezpečnost celého řešení.

Bude se tedy orientovat na ty oblasti, které jsou často upozaďovány z příčin popsaných v kontextu problematiky. Návrh nového objektově orientovaného webového frameworku bude vycházet z analýzy dané oblasti prostřednictvím rozboru populárních a široce používaných projektů s ohledem na jejich strukturu a uspořádání. Výsledky implementace budou ověřeny především na ukázce reálného použití a názorným srovnáním s analyzovanými webovými frameworky v oblasti časových a paměťových nároků.

Kapitola 2

Základní pojmy

Tato kapitola se věnuje definicím a popisům základních pojmů, které sehrají velmi důležitou roli při rozboru problematiky webových frameworků a poté i v samotném návrhu vlastního řešení.

2.1 Softwarový framework

Framework je sada softwaru, která organizuje architekturu aplikace a usnadňuje práci vývojáře [3]. Naopak Stanojević a kolektiv jej definují méně obecně jako kostru aplikace, jež obsahuje kompletní kód pro základní funkce systému, který lze přizpůsobit potřebám konkrétní vyvíjené aplikace [4]. Z těchto definic vyplývá, že jejich využití napříč spektrem aplikací používaných v různých odvětvích může být velmi značné, jelikož nabízejí zčásti hotová řeření v dané problematice, která mají přímý vliv nejen na výsledný produkt,
ale i na celkový průběh vývoje.

2.2 Webový framework

Webový framework je specifický případ softwarového frameworku, jež obsahuje sadu tříd a funkcí, která je mnohdy doplněna dalšími knihovnami
a programy, jejichž hlavním účelem je poskytnout otestovaná řešení pro opakovaně se vyskytující problémy a tím urychlit celkový vývoj webové aplikace. Lze jej také definovat jako vývojové rozhraní, sadu softwarových nástrojů pro vytváření dynamických webových stránek, aplikací a služeb, které jsou specifickou realizací abstraktního návrhového vzoru [5].

2.3 Obecné členění

Webové frameworky lze elementárně rozčlenit do dvou kategorií, serverové a klientské. Jejich primární charakteristika je prostředí, ve kterém pracují, což má zásadní vliv na jejich fungování a účel.

Pokud webové framewroky běží na straně klienta, tak zpravidla reagují na uživatelský vstup a upravují koncový vzhled stránky, avšak v rámci webových prohlížečů může dojít k nekonzistentnímu chování, jelikož není zaručena podpora nejnovějších standardů a také mohou být ovlivněny nainstalovanými doplňky.

V případě serveru lze celkový zdrojový kód optimalizovat pro určitý hardware a tím zaručit jeho spolehlivost. Na této straně se webové frameworky starají především o obsluhu uživatelských požadavků a práci s daty, která jsou zpravidla ukládána do logických struktur.

2.4 CGI

CGI je zkratkou pro Common Gateway Interface, což je standardní protokol pro spouštění programů na webovém serveru [6]. Za pomoci tohoto protokolu lze využít externích aplikací či skriptů jako nástroj pro obsluhu uživatelských dotazů a zpracování uživatelských dat. Jejich předávání probíhá především za pomoci proměnných prostředí (anglicky *environment variables*) a standardního vstupu. Přesné chování bylo standardizováno vývojáři z The Apache Software Foundation pod označením RFC3875 [7].

2.5 Kybernetická bezpečnost

Kybernetická bezpečnost je soubor nástrojů, zásad, bezpečnostních konceptů, bezpečnostních záruk, pokynů, rizik, manažerských přístupů, akcí, školení, osvědčených postupů, ujištění a technologií, které lze použít k ochraně kybernetického prostředí, organizace a majetku [8]. Jejím účelem v prostředí internetu je zabránit především neoprávněnému přístupu k citlivým datům, zdrojovým kódům a použitým technologiím.

Kapitola 3

Analýza problematiky

Tato kapitola se věnuje analýze problematiky webových frameworků. Z velké části budou rozebrána již existující řešení, která si prošla dlouhodobým evolučním vývojem a jsou v prostředí internetu běžně používána. V případě diference využitých technologií některých částí, jíž může být například programovací jazyk, dojde k jejich srovnání.

3.1 Vymezení analýzy

Z důvodu existence standardního protokolu CGI není nutné klást limity na pogramovací jazyk, který bude použit při tvorbě webové aplikace.

Avšak v posledních letech se pro malé a středně velké projekty začal využívat jazyk PHP, který je velmi často součástí cenově dostupných webových hostingů. Současně v tomto prostředí začaly vznikat webové frameworky, mezi které patří například české Nette, které je také využíváno významnými společnostmi ve střední Evropě [1]. Naopak ve světě se stal rozšířeným Laravel, který sám sebe definuje jako framework pro webové aplikace s expresivní a elegantní syntaxí [2].

S rostoucí popularitou jazyka Python začaly vznikat různé projekty, které by umožňovaly jeho využití v internetovém prostředí. Mezi nejrozšířenější řešení lze bezpochyby zařadit bezplatný webový framework Django, který je implementovaný s důrazem na čistotu a pragmatičnost [9].

Z prostředí systémových jazyků pochází CppCMS, který je vyvíjen zejména Artyomem Beilisem již několik let. Ač má ve svém názvu CMS, což je anglická zkratka pro *content management system* a značí komplexní systém pro správu obsahu [10], jedná se skutečně o webový framework, který se zaměřuje především na rychlost. Tomuto projektu se nedostalo takové podpory komunity jako v předchozích případech a je stále vyvíjen pouze jedinci, proto je zde zmíněn především z historického hlediska a nebude do rozboru v této práci zařazen.

Na poli internetu existuje mnoho dalších řešení, která si svojí kvalitou zpracování zasluhují být součástí analýzy, avšak pro potřeby získání náhledu do jejich struktury bude omezena pouze na výše zmíněné implementace.

3.2 Metody analýzy

Analýza problematiky se bude skládat ze studia dostupných oficiálních dokumentací jednotlivých webových frameworků, porozumění programově vícejazyčnému zdrojovému kódu, ověřování a doplňování informací z odborných publikací či literatury, popisu souvisejících mezinárodních standardů, hledání společných rysů, které mohou být nápomocné při návrhu nového webového frameworku, a seznámení čtenáře s výsledkem rozboru dílčích částí.

3.3 Cílová skupina

Jak již bylo napsáno ve výše uvedené definici webového, ale i obecného frameworku, cílem je především zjednodušit a urychlit vývoj aplikací.
Z těchto důvodů cílovou skupinou všech analyzovaných projektů nejsou pouze korporátní společnosti, organizace či jedinci, ale kdokoli, kdo chce čerpat přínosy takového řešení.

3.4 Licence

Licence se stává základním klíčem k úspěchu každého softwaru. Filozofií analyzovaných webových frameworků je odepření možnosti vztažení práv pouze na jeden subjekt, který by si tím zajistil výhradní použití. Za tímto účelem jsou zdrojové kódy dostupné pod svobodnými licencemi, které tuto filozofii zachovávají [11].

Nette je nabízen pod GPLv2, GPLv3 a New BSD License, aby v případě uplatnění s jiným softwarem zajistil dostatečnou právní kompatibilitu [1]. Naopak projekty Laravel [11] a Django [12] se vydaly cestou svobodnější licence MIT a New BSD License, které kladou minimální podmínky na užití a tím umožňují kombinaci se softwarem distribuovaným pod restriktivní licencí (viz tabulka 3.1).

Framework	Licence
Nette	New BSD License, GPL v2/v3
Laravel	MIT
Django	New BSD License

Tabulka 3.1: Přehled licencí analyzovaných webových frameworků.

3.5 Životní cyklus dotazu

Porozumění životnímu cyklu dotazu a odpovědi na něj je velmi důležité pro pochopení principu daného nástroje jako takového. Webové aplikace běžně používají pro komunikaci s okolním světem protokoly HTTP nebo HTTPs [13], jež mají specifickou strukturu, na kterou je nutné brát ohledy především při odesílání odpovědi. Podrobné rozebrání konceptu fungování jednotlivých webových frameworků, tedy jejich životních cyklů, umožní vyčlenit podstatné komponenty pro tuto analýzu.

3.5.1 Laravel

Webové servery většinou považují soubory s názvem `index` jako hlavní.
V případě serverové konfigurace ve webovém frameworku Laravel tomu není jinak. Hlavní soubor s příponou `php` je uložen ve veřejném adresáři. Kromě automatického načítání některých komponent se zde vytváří samotná instance aplikace Laravel, která je využita k vytvoření `Kernel` objektu. Tomuto jádru je předán objekt `Request`, jež obsahuje všechny podstatné informace o HTTP požadavku[1] (viz zdrojový kód 3.1).

```php
use Illuminate\Contracts\Http\Kernel;
use Illuminate\Http\Request;

define('LARAVEL_START', microtime(true));

if (file_exists(__DIR__.'/../storage/framework/maintenance.php')) {
    require __DIR__.'/../storage/framework/maintenance.php';
}

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

$app = require_once __DIR__.'/../bootstrap/app.php';

$kernel = $app->make(Kernel::class);

$response = $kernel->handle(
    $request = Request::capture()
)->send();

$kernel->terminate($request, $response);
```

Obrázek 3.1: Výchozí indexový soubor frameworku Laravel

Uvnitř jádra se především vytvoří instance všech poskytovatelů služeb, které jsou zadefinovány v konfiguračním souboru, a u každého z nich se zavolá

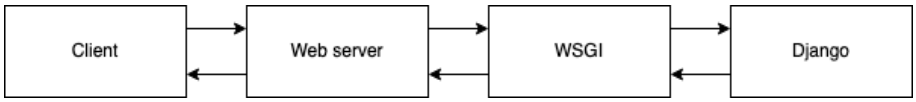
[1] viz zdrojový kód Laravel: https://github.com/laravel/laravel/tree/8.x/public

metoda registrace. Jakmile se všichni poskytovatelé zaregistrují, jádro je spustí. Jejich úloha je důležitá, jelikož jsou zodpovědni za načítání různých komponent samotného frameworku.

Po registraci všech poskytovatelů služeb je HTTP požadavek předán směrovači, což je další poskytovatel služeb, který jej pošle dále ke zpracování, zpravidla do řadiče (anglicky *controller*). Zde se provedou konkrétní úkony zadefinované vývojářem a vrátí se odpověď tak, aby aplikace Laravel měla možnost její úpravy, a poté se odešle uživateli[2].

3.5.2 Django

Při používání frameworku Django bude většinou využito Nginx/uWSGI/Django, Apache/mod_wsgi/Django nebo jiné podobné softwarové kombinace. Kromě webového serveru a Django je zde také použita variace WSGI, což je zkratka pro Web Server Gateway Interface. Jak již název napovídá, jedná se o rozhraní mezi webovým serverem a aplikací, respektive mezi webovým serverem a Django (viz obrázek 3.2).



Obrázek 3.2: Životní cyklus dotazu mezi klientem a Django

Každý příchozí HTTP požadavek je zpracováván funkcemi `middleware`, které jsou zadefinovány v souboru obsahující nastavení. Každý `middleware` je zodpovědný za něco jiného. Je možné vytvářet nová, ale i využít předdefinovaná řešení obsažená přímo ve webovém frameworku. Jejich funkcionalitu lze připodobnit poskytovatelům služeb v implementaci Laravel.

Poté, co proběhnou předchozí kroky, aplikace převezme z požadavku dotazovanou URL adresu. Součástí konfigurace je odkaz na modul, jehož součástí jsou vzory jednotlivých adres, jež jsou provázány s pohledem (anglicky *view*). Jakmile bude nalezena shoda mezi vzorem a URL adresou, směrovač odešle požadavek do příslušného pohledu.

Po zpracování dat v pohledu je požadavek poslán ke kontextovým procesorům, jež generují pomocná data pro správné vykreslení šablon a HTTP odpovědi. Závěrem je požadavek i s odpovědí znovu předán funkcím `middleware`, které informace zpracují, a výsledek se odešle uživateli.

3.5.3 Nette

V Nette, jak tomu je i v implementaci webového frameworku Laravel, je vyčleněn veřejný adresář obsahující hlavní soubor `index.php`, do kterého

[2] viz dokumentace Laravel: https://laravel.com/docs/8.x/lifecycle

směřují všechny příchozí požadavky. Zde se na počátku vytvoří pracovní prostředí, jež například obsahuje nástroje na zaznamenávání a odchytávání chyb generovaných v průběhu celé aplikace. Z tohoto prostředí je vyčleněn `container`, jehož hlavní náplní je generování objektů dle zadaných parametrů. Za použití metody kontejneru je vytvořen objekt aplikace, který je následně v dalším kroku spuštěn[3] (viz zdrojový kód 3.3).

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

$configurator = App\Bootstrap::boot();
$container = $configurator->createContainer();
$application = $container->getByType(Nette\Application\Application::class);
$application->run();
```

Obrázek 3.3: Výchozí indexový soubor frameworku Nette

Aplikace je složena z presenterů, jež je možné částečně chápat jako obdobu kontrolérů. Pro zjištění, jaký presenter je zodpovědný za vyřízení takového typu požadavku, požádá směrovač, který o tom rozhodne na základě dostupných dat a své konfigurace. Jakmile je znám příslušný presenter, aplikace požádá kontejner o jeho vytvoření. Samotné vyřízení požadavku probíhá uvnitř presenteru, který vrací odpověď, jež nemusí být přímo HTML stránka, ale i jiný datový typ jako je například XML, JSON a obrázek[4].

3.6 HTTP požadavek

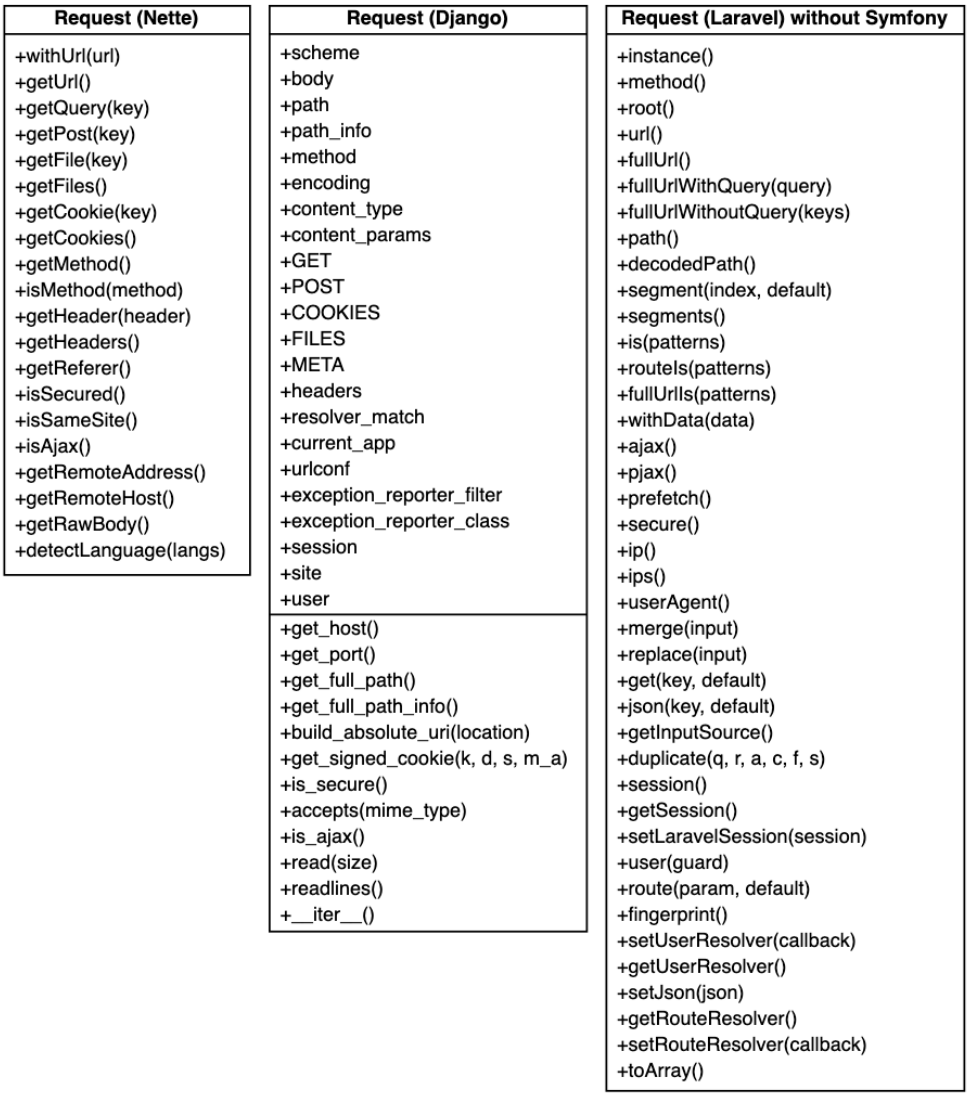
Z úvodních popisů životních cyklů webových frameworků Nette, Laravel a Django vyplývá, že shodně na počátku svého spuštění za pomoci předpřipravených objektů zpracovávají HTTP požadavek, jež obsahuje všechny podstatné informace o dotazu [14]. Jak již bylo popsáno v sekci o Common Gateway Interface (2.4), webový server všechna dostupná data předá samotné aplikaci ve formě proměnných prostředí. V případě jazyka PHP jsou data pro vývojáře zpracována do super globálních proměnnných.

Struktura a princip tříd zpracovávajících HTTP požadavek jsou ve všech případech analyzovaných webových frameworků velmi podobné. Shodně přijímají proměnné obsahující informace o uživateli, cookies, zaslaných souborech, URL adrese a datech z formulářů. Vše vnitřně zpracovávají do jednotlivých proměnných dané třídy a mají naimplementované metody, které při svém zavolání vrací konkrétní informace (viz obrázek 3.4). Velmi podstatnou vlastností je, že všechny atributy by měly být ve stavu pouze pro čtení. Tento rys zaručuje konzistenci dat v průběhu celého procesu odbavení dotazu na serveru.

[3] viz zdrojový kód Nette: https://github.com/nette/web-project/tree/master/www
[4] viz dokumentace Nette: https://doc.nette.org/cs/3.1/application

Rozdíly mezi jednotlivými řešeními je možné spatřit v možnostech dotazování se jednotlivých informací. Například projekt Laravel se rozhodl vydat cestou maximálního vytěžování dat z HTTP požadavku. Za tímto účelem je použito již hotové řešení z webového frameworku Symfony, které je rozšířené o další funkcionality[5]. Tuto variantu kombinování zdrojového kódu umožňuje především stejný programovací jazyk PHP a přívětivá licenční politika obou řešení.

Velmi podobnou filozofii přístupu k informacím zvolili vývojáři projektu Django. Oproti předchozímu webovému frameworku Laravel je možné k některým datům přistupovat přímo bez použití metod, aniž by byla ohrožena jejich konzistence[6].



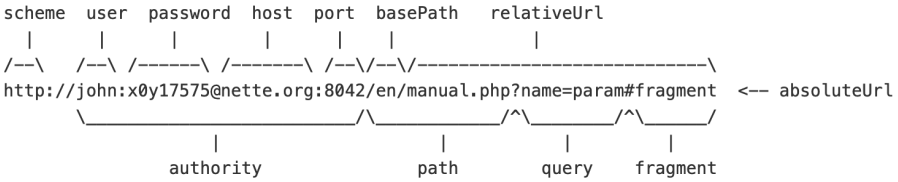
Obrázek 3.4: Atributy a základní přístupné metody analyzovaných request tříd

[5] viz dokumentace Laravel: https://laravel.com/docs/8.x/requests
[6] viz dokumentace Django: https://docs.djangoproject.com/en/3.2/ref/request-response/

Avšak jednodušší implementace, která řeší pouze základní problematiku bez pokročilejších funkcionalit, je při svém běhu méně náročná na serverové zdroje, což má pozitivní vliv při velkém zatížení. Vývojáři frameworku Nette se vydali spíše minimalistickou cestou a ve třídě zpracovávající HTTP požadavek připravili pouze základní funkcionality, které jsou dostatečně schopné vyhovět v základních požadavcích[7].

3.7 URL adresa

Správné zpracování URL adresy dotazu je klíčové pro budoucí rozhodování aplikace, která později na jejím základě vytvoří objekt či zavolá funkci, jež bude vykonávat konkrétní logiku nad samotným požadavkem. Za účelem unifikovaného chování napříč spektrem webových prohlížečů byl vytvořen mezinárodní standard pod kódovým označením RFC3986 (viz obrázek 3.5), který mimo jiné přímo stanovuje strukturu každé URL adresy [15].



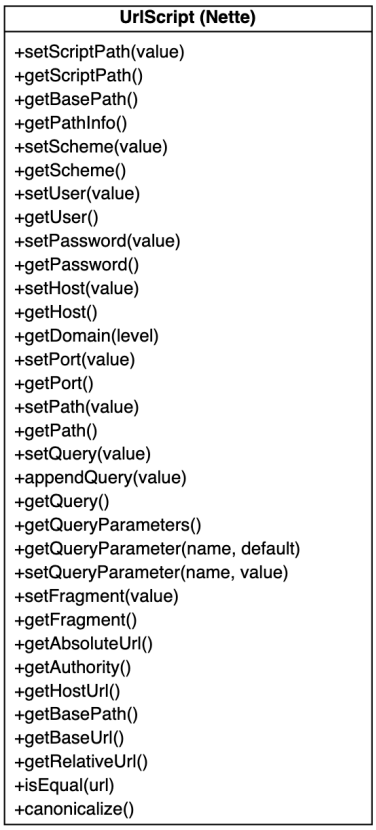
Obrázek 3.5: URI syntax dle RFC3986 [1]

Důležitost absolutní URL adresy spočívá ve množství informací, které jsou v ní obsažené. Rozhodne-li se uživatel například pro komunikaci za užití protokolu HTTP, je možné pokusit se jej přesměrovat na zabezpečený komunikační kanál HTTPs, který zaručí bezpečný přenos zaslaných dat. Lze také zjistit požadovanou jazykovou mutaci, je-li uvedena, dotaz či název konkrétní stránky ve webové aplikaci a mnoho dalších informací.

Přístupy jednotlivých webových frameworků k této problematice se do značné míry liší, jelikož samotné informace o dotazované URL adrese jsou obsaženy již ve třídě zpracovávající HTTP požadavek.

Nette framework z českého prostředí se oproti jiným analyzovaným řešením vydal cestou relativně podrobného zpracování dat oddělených do jedné samostatné třídy (viz obrázek 3.6), kde implementované metody umožňují získávání kterékoliv části URL adresy.

[7] viz dokumentace Nette: https://doc.nette.org/cs/3.1/http-request-response



Obrázek 3.6: Základní přístupné metody UrlScript třídy v Nette [1]

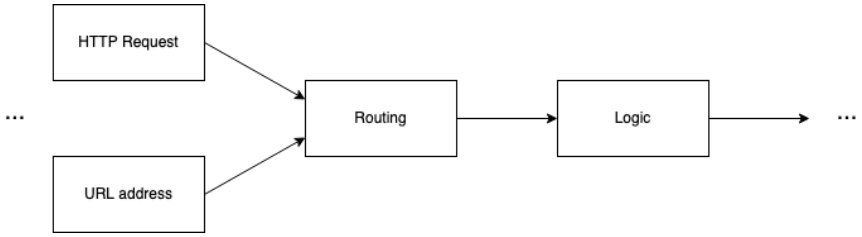
Python vývojáři se v projektu Django rozhodli zachovat informace o URL adrese jako součást třídy, která obsahuje data o HTTP požadavku a vytvořili pomocné nástroje k jejich zpracování především do atributů `path` a `path_info` (viz obrázek 3.4). Oproti předchozímu řešení v Nette je možné všechny podstatné informace nalézt v atributech jedné třídy, avšak z důvodu rozsáhlosti obsažených informací se nejedná o tak přehledné řešení[8].

Jak již bylo zmíněno výše, s architekturou Laravel je značně provázána třída zpracovávající HTTP požadavek z webového frameworku Symfony. Společně vytváří velmi dobré a především rozsáhlé rozhraní pro dotazování se jednotlivých informací z URL adresy bez nutnosti vytváření dalšího objektu[9]. V téže třídě jsou mimo jiné naimplementovány metody umožňující sledovat pohyb a historii navštívených URL adres v dané aplikaci jednotlivých uživatelů, což je vhodné pro identifikaci při vícenásobné komunikaci se serverem.

[8] viz dokumentace Django: https://docs.djangoproject.com/en/3.2/ref/request-response/
[9] viz dokumentace Laravel: https://laravel.com/docs/8.x/requests

3.8 Směrování

Statická webová stránka, jež je zpravidla tvořena HTML dokumenty, je uživateli odeslána přesně tak, jak je na serveru uložena a k jednotlivým souborům se přistupuje za pomoci absolutní URL adresy. Avšak tento přístup v dynamicky generovaných webových aplikacích není možný, jelikož na dotazované URL adrese neexistuje žádný soubor. Webový framework musí nejdříve zjistit, jakou konkrétní logiku pro generování odpovědi nad samotným požadavkem má vykonat. Toto rozhodnutí se v analyzovaných řešeních zakládá na principu směrování, anglicky *routing*, který na základě zpracovaných dat z HTTP požadavku a URL adresy rozhodne (viz obrázek 3.7).



Obrázek 3.7: Schéma umístění směrování uvnitř webového frameworku

Ve webovém frameworku Nette je k dispozici třída `RouteList`[10], která realizuje princip routování. Uvnitř tohoto objektu jsou zadefinány směry za pomocí URL masky, názvu presenteru (viz návrhové vzory 3.9) a názvu akce. Pořadí zadefinování je velmi důležité, jelikož se v případě rozhodování, jaký presenter a jakou akci zavolat, postupuje odshora dolů. Běžnou součástí projektů založených na Nette je továrna, která při svém zavolání vrací objekt třídy `RouteList` (viz obrázek 3.11), jež je naplněn všemi cestami a je připraven na okamžité použití. Za velmi důležitou vlastnost lze považovat obousměrnost zpracování dat. Umožňuje nejen z URL adresy získat informace, ale i z informací získat URL adresu (viz zdrojový kód 3.8).

```
class RouterFactory {
    public static function createRouter(): RouteList {
        $router = new RouteList;
        $router->addRoute('<presenter>/<action>', Homepage:default);
        $router->addRoute('products', 'Products:default');
        $router->addRoute('product/<id>', 'Product:view');

        return $router;
    }
}
```

Obrázek 3.8: Ukázka Továrny objektu RouteList v Nette

[10] viz dokumentace Nette: https://doc.nette.org/cs/3.1/routing

V řešení Laravel jsou všechny směry definovány v samostatném souboru `routes/web.php` (viz zdrojový kód 3.9), které jsou načteny poskytovatelem služeb z jádra aplikace. Pokud je Laravel využit k vytvoření API, což je zkratka pro *application programming interface*, je pro směry v adresáři `routes` připraven soubor `api.php`, který zajišťuje, že se k nim bude v rámci povahy aplikace přistupovat vhodněji. Při definici každé nové položky je zadána akceptující HTTP metoda požadavku, akce a URL adresa s parametry, které mohou být omezeny regulárními výrazy. Pokud bude daný směr pojmenován metodou `name()`, je možné za pomocných funkcí reverzně sestavit jeho původní URL adresu obdobně jako v implementaci Nette[11].

```
use Illuminate\Support\Facades\Route;

Route::get('/', function () {return view('Hello world!');});
Route::get('/user/profile', [UserProfileController::class, 'show']);
Route::get('/home', [HomeController::class, 'showHome'])->->name('home');
```

Obrázek 3.9: Ukázka struktury souboru web.php v Laravel

Django dle příchozího HTTP požadavku zpracovaného v objektu třídy `HttpRequest` nebo výchozího nastavení načte modul obsahující směry. Všechna data by měla být uložena v poli `urlpatterns`, jež bude obsahovat informace o směrech uložené ve formátu sekvence instancí `django.urls.path()`[12], které obsahují informaci o vzoru URL adresy a pohledu (viz zdrojový kód 3.10). Jednotlivé položky jsou procházeny odshora. Jakmile bude nalezena shoda, Django importuje pohled (viz návrhové vzory 3.9) a předá mu instanci objektu třídy `HttpRequest` a další informace.

```
from django.urls import path

from . import views

urlpatterns = [
    path('product/2003/', views.special_product_2003),
    path('product/<int:id>/', views.product),
    path('product/<int:id>/<id:subid>/', views.subproduct)
]
```

Obrázek 3.10: Ukázka modulu obsahující URL vzory v Django

[11] viz dokumentace Laravel: https://laravel.com/docs/8.x/routing
[12] viz dokumentace Django: https://docs.djangoproject.com/en/3.2/topics/http/urls/

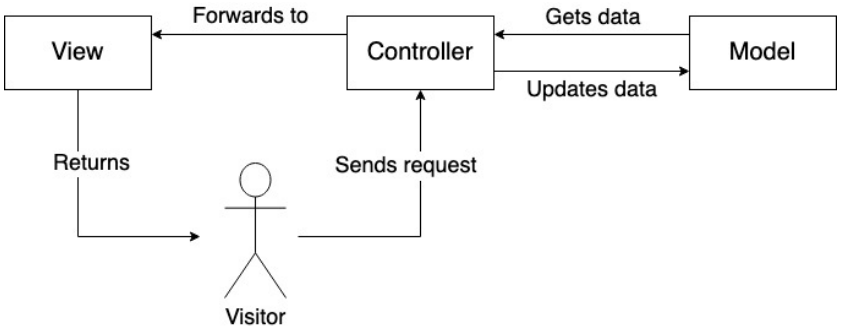
Obrázek 3.11: Základní přístupné metody routovacích tříd Nette a Laravel [1, 2]

3.9 Návrhové vzory

V předchozích částech analýzy bylo často odkazováno na konkrétní logiku, která se má nad dotazem vykonat. Také byly zmíněny pojmy jako *controller*, *presenter* a *view* v kontextu jednotlivých webových frameworků. Právě tato logika je mnohdy vyčleněna právě do tříd nebo souboru funkcí, jež nesou tyto názvy.

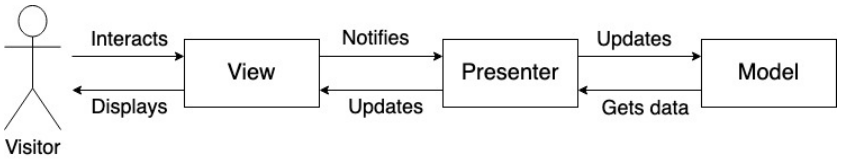
Gamma a kolektiv definuje návrhové vzory v knize *Design Patterns: Elements of Reusable Object-Oriented Software* jako popisy komunikujících objektů a tříd, které jsou přizpůsobeny tak, aby řešily obecný návrhový problém v konkrétním kontextu [16]. V prostředí webových aplikací nabízejí ucelený princip programování a v závislosti na architektuře rozdělují datový model, uživatelské rozhraní a řídicí logiku do oddělených komponent tak, aby měly minimální vliv na ostatní.

Mezi základní návrhové vzory lze zařadit *Model-View-Controller* (MVC), který právě separuje výše zmíněné části (viz obrázek 3.12). V internetovém prostředí jej lze nalézt například v aplikacích, které jsou postavené na technologii Laravel [17].



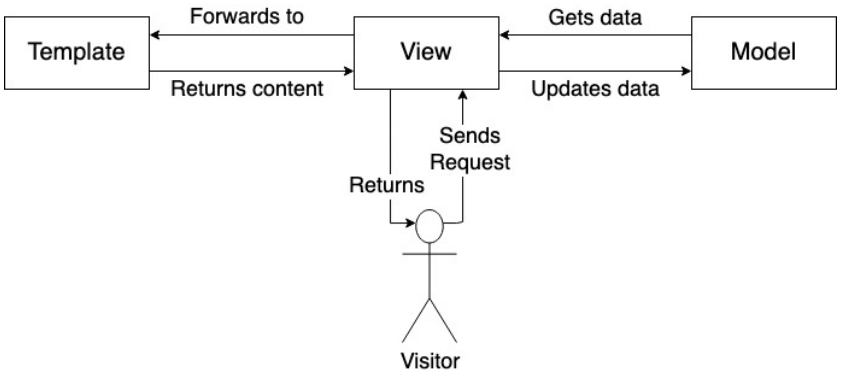
Obrázek 3.12: Diagram MVC architektury

V současnosti existuje mnoho podobných architektur, které vznikly za stejným účelem. Primárně se liší ve vstupním bodě uživatelského dotazu a jeho životním cyklem v jednotlivých komponentách. Příkladem lze uvést *Model-View-Presenter* (viz obrázek 3.13), který je podporován ve webovém frameworku Nette [1].



Obrázek 3.13: Diagram MVP architektury

Naopak v případě projektu Django se vývojáři rozhodli pro méně běžnou architekturu nazvanou *Model-Template-View* (MTV) [18]. V případě předcházejících architektur nebylo explicitně řečeno, jakým způsobem má být uživatelské rozhraní implementováno. Jak již název sám o sobě napovídá, pro vzhledovou část bude využito šablon, které budou doplněny daty z modelu a zobrazeny třídou *View* (viz obrázek 3.14).



Obrázek 3.14: Diagram MTV architektury

3.10 Databáze

Komponenta modelu je součástí všech analyzovaných návrhových vzorů a je zodpovědná za správu dat. To může zahrnovat propojení s jakýmkoliv trvalým úložištěm dat a používání datových struktur založených na paměti k ukládání dat během provádění. Komponenta modelu implementuje logiku, která umožňuje aplikaci vytvářet, číst, aktualizovat a odstraňovat data aplikace [19].

Mezi nejrozšířenější řešení pro ukládání dat v modelech lze jednoznačně zařadit MySQL databázi, která zaručuje bezpečný paralelní přístup do jednotlivých tabulek a jednoduchou správu dat za použití jazyka SQL (viz ukázka 3.15), což je zkratka pro *Structured Query Language*. Kromě MySQL existují i jiná řešení, jako je například MariaDB, PostgreSQL, SQLite nebo Microsoft SQL Server, která používají stejný dotazovací jazyk, ale vnitřně se v principech provádění jednotlivých úkonů částečně liší.

```sql
INSERT INTO CUSTOMERS
(FirstName, LastName, Street, City, Country, Zipcode, Phone)
VALUES
('John', 'Blazek', '191 Loco Ave.', 'El Sacra', 'CA', '92683', '420 432-5691')
```

Obrázek 3.15: Ukázka jazyka SQL

Laravel nabízí jednotné a jednoduché rozhraní (viz zdrojový kód 3.16) pro širokou škálu databázových systémů, které lze konfigurovat v příslušném souboru `config/database.php`. Kromě unifikovaného řešení se vývojáři zaměřili na ochranu dat a příslušné metody implementovali s důrazem na bezpečnost, aby maximálně předešli `SQL injection` a dalším bezpečnostním hrozbám[13].

```php
// knihy z tabulky book
$books = DB::select('select * from books');

foreach ($books as $book) {
    echo 'title:  ' . $book->title;
    echo 'author: ' . $book->author;

    echo 'book categories: ';
    $categories = DB::select('select * from book_categories where book_id = ?', [$book->id]);
    foreach ($categories as $bookCategory) {
        // $bookCategory je řádek z tabulky 'book_categories'
        echo $bookCategory->category->name . ', ';
    }
}

$books = DB::select('select * from books where active = ?', [1]);
```

Obrázek 3.16: Ukázka databázového SQL dotazu ve frameworku Laravel

[13] viz dokumentace Laravel: https://laravel.com/docs/8.x/database

Ve webovém frameworku Nette lze kromě základní vrsty pro přístup do databéze nalézt `Nette Database Explorer`, což je rozhraní, které umožňuje získávat data z databáze zjednodušeným způsobem bez nutnosti hlubší znalosti SQL jazyka (viz zdrojový kód 3.17). Cílem tohoto rozhraní je pokládat dotazy efektivně bez přenášení zbytečných dat. Ačkoli se v některých případech nevyrovná jazyku SQL, jedná se o velmi pokročilý nástroj usnadňující alespoň elementární práci s databází[14].

```php
// knihy z tabulky book
$books = $explorer->table('book');

foreach ($books as $book) {
    echo 'title:  ' . $book->title;
    echo 'author: ' . $book->author;

    echo 'book categories: ';
    foreach ($book->related('book_categories') as $bookCategory) {
        // $bookCategory je řádek z tabulky 'book_categories'
        echo $bookCategory->category->name . ', ';
    }
}
```

Obrázek 3.17: Ukázka databázového SQL dotazu ve frameworku Nette

Jazyk PHP nativně obsahuje nástroje pro práci s databází, ze kterých mohou webové frameworky Laravel a Nette těžit [20]. Avšak toto neplatí pro projekt Django, který je implementovaný v jazyce Python. Dle dokumentace jsou oficiálně podporovány pouze PostgreSQL, MariaDB, MySQL, Oracle a SQLite. V případě nedostatečné podpory, kterou implementace nabízí, je nutné použít řešení třetích stran, což je plně v souladu s doporučením a filozofií vývojářů webového frameworku Django[15].

3.11 HTTP odpověď

Před všechna data, která jsou uživateli odeslána za užití protokolu HTTP, musí být vložena hlavička, jež obsahuje doprovodné informace o stránce a popřípadě návodné parametry, jak s ní v rámci prohlížeče zacházet (viz obrázek 3.18). Všechny náležitosti včetně povinných a nepovinných parametrů jsou popsány v mezinárodním standardu RFC2616, ke kterému přispívali významné osobnosti a společnosti z prostředí internetu [21].

[14] viz dokumentace Nette: https://doc.nette.org/cs/3.1/database
[15] viz dokumentace Django: https://docs.djangoproject.com/en/3.2/ref/databases/

```
HTTP/1.1 200 OK
Date: Sun, 28 Nov 2021 16:55:07 GMT
Server: Apache/2.4.46 (Unix)
Strict-Transport-Security: max-age=31536000; includeSubDomains
Expect-CT: max-age=86400, enforce
X-Frame-Options: deny
X-Content-Type-Options: nosniff
Cache-Control: no-cache, must-revalidate
Vary: User-Agent
X-XSS-Protection: 1; mode=block
Content-Security-Policy: script-src 'self' 'unsafe-inline';
X-Permitted-Cross-Domain-Policies: none
Feature-Policy: vibrate 'none'; geolocation 'none'
X-Powered-By: ANNMu
Last-Modified: Sun, 28 Nov 2021 16:55:07 GMT
Keep-Alive: timeout=5, max=99
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html; charset=utf-8
```

Obrázek 3.18: Ukázka hlavičky HTTP odpovědi

V případě webového frameworku Nette jsou všechna data o HTTP hlavičce ukládána do samostatného objektu, jež je dostupný v každém presenteru (viz zdrojový kód 3.19). Oproti třídě zpracovávající HTTP požadavek jsou všechny informace v třídních proměnných přepsatelné, avšak pouze do okamžiku prvního odeslání dat na standartní výstup. Pro lepší přehlednost a především srozumitelnost zdrojového kódu jsou pro stavové kódy odpovědi předdefinované konstanty, které nevyžadují přímou znalost číselného označení[16].

```
$httpResponse = $this->getHttpResponse()

$httpResponse->setCode(Nette\Http\Response::S404_NOT_FOUND)

$httpResponse->setHeader('Pragma', 'no-cache');
$httpResponse->setHeader('Accept', 'application/json');
```

Obrázek 3.19: Ukázka práce s objektem pro HTTP odpověď v Nette

Opačný přístup lze najít v projektu Django. Na rozdíl od objektu zpracovávající HTTP požadavek, který vzniká automaticky při spuštění aplikace, je objekt zpracovávající HTTP odpověď plnou zodpovědností vývojáře. Každý vytvořený pohled je zodpovědný za vytvoření instance třídy `HttpResponse`, naplnění a její vrácení[17]. Součástí `HttpResponse` jsou také předpřipravené podtřídy, které zpracovávají ruzné typy odpovědí s různými stavovými kódy, a tím zpřehledňují celkové řešení (viz zdrojové kódy 3.20 a 3.21).

[16] viz dokumentace Nette: https://doc.nette.org/cs/3.1/http-request-response
[17] viz dokumentace Django: https://docs.djangoproject.com/en/3.2/ref/request-response/

```
return HttpResponseRedirect('/contact/thanks/')
```

Obrázek 3.20: HTTP odpověď za užití podtřídy v Django

```
response = HttpResponse()
response.status_code = 302
response.headers['Location'] = '/contact/thanks/'
return response
```

Obrázek 3.21: HTTP odpověď bez užití podtřídy v Django

Každý směr, jež používá komponenty řešení Laravel, by měl vrátit odpověď, která je odeslána do prohlížeče uživatele. Zde není striktní povinnost vývojáře vracet přímo objekt obsahující informace o hlavičce. Pokud aplikace Laravel obdrží pouze řetězec složený výhradně z těla stránky, automaticky jej doplní o výchozí HTTP hlavičku a v modifikované verzi odešle. Ačkoli je tento přístup velmi přívětivý, obvykle je nutné mít kontrolu nad úplným obsahem dat. Pro tuto variantu je k dispozici třída `Response` (viz zdrojový kód 3.22), která je silně provázaná s třídou `Symfony\Component\HttpFoundation\Response` z webového frameworku Symfony, jako tomu je v třídě zpracovávající HTTP požadavek[18].

```
return response($content)
        ->header('Content-Type', $type)
        ->header('X-Powered-by', 'Laravel')
        ->header('ETag', '0815');
```

Obrázek 3.22: HTTP odpověď ve frameworku Laravel

3.12 Shrnutí

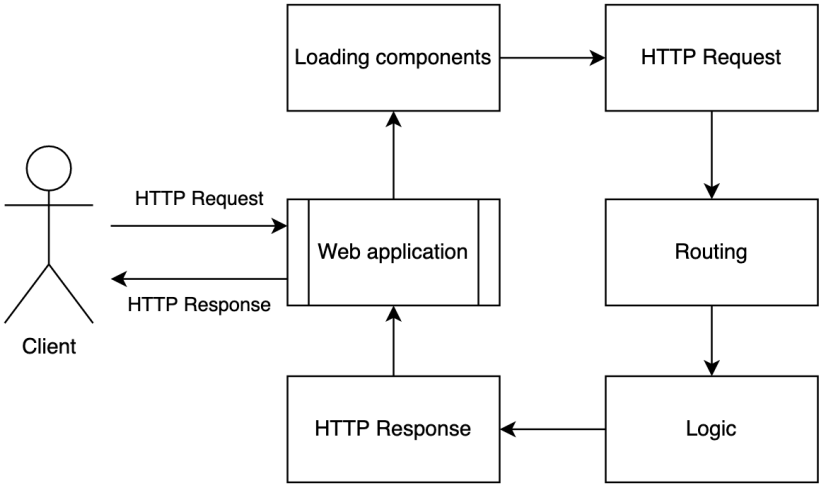
Hlavním cílem analýzy vybraných webových frameworků byla identifikace společné obecné struktury jednolivých řešení, získání základního pohledu do vnitřích procesů, jež se provádějí automaticky při každém spuštění aplikace, a nalezení často používaných komponent, které zpřehledňují či usnadňují vývoj zásadním způsobem.

Provedením podrobné analýzy webových frameworků Nette, Laravel a Django za použití oficiálně publikovaných dokumentací dostupných v internetovém prostředí, zdrojových kódů, mezinárodně uznávaných standardů a odborné literatury nejen v českém, ale i anglickém jazyce, byly identifikovány

[18] viz dokumentace Laravel: https://laravel.com/docs/8.x/responses

a jednotlivě popsány jejich průběhy od počátečního spuštění aplikace až do závěrečného okamžiku odeslání odpovědi uživateli.

Na počátku svého životního cyklu (viz obrázek 3.23) jsou automaticky vytvářeny objekty, které jsou nezbytné pro správné fungování následujících částí programu. Principiálně se implementace v jednotlivých řešení příliš neliší, pouze nesou jiný název. Zatímco například v dokumentaci Django lze nalézt pojmenování `middleware`, tak v popisu aplikace Laravel je možné nalézt termín *poskytovatelé služeb*.



Obrázek 3.23: Zjednodušený životní cyklus analyzovaných webových frameworků

HTTP požadavky jsou ve všech případech analyzovaných webových frameworků zpracovány do samostatných objektů, jejichž součástí jsou metody poskytující podrobnější informace o dotazech.

Dalším společným rysem, jež vyplývá z analýzy, je princip směrování. Po dokončení zpracování dat a inicializování objektů je nutné rozhodnout o následující logice, která se má vykonat. Zde je idea způsobu provedení v jednotlivých řešeních velmi podobná. Na základě zadefinovaného vzoru URL adresy a popřípadě metody dotazu je rozhodnuto, která anonymní funkce, kontrolér, presenter či pohled bude zavolán.

Konkrétní logika může mít mnoho podob. V dnešní době jí dominují návrhové vzory, které nabízí ucelené principy programování a řešení dílčích problémů. Jejich hlavním cílem je oddělení datového modelu, uživatelského rozhraní a řídící logiky tak, aby na sebe navzájem měly minimální vliv, což má pozitivní efekt především ve velkých projektech. Postupně byly popsány architektury *Model-View-Controller* (MVC), *Model-View-Presenter* (MVP) a *Model-Template-View* (MTV), tedy pro každý webový framework jiný návrhový vzor. Objektivně nelze zhodnotit, který je výhodnější, jelikož všechny mají stejný účel a podobný koncept.

V závěru životního cyklu se analyzovaná řešení vypořádávají s podobou HTTP hlavičky odpovědi. Zatímco v projektu Django je tento úkon povinností vývojáře, v Nette či Laravel tomu tak úplně není. Jejich součástí je výchozí HTTP hlavička pro odpovědi, která bude použita v případě, že nedojde ke změně. Její podoba má zásadní vliv na bezpečnost, a proto by měla být zadefinována s maximální precizností.

Popis a rozbor fungování vybraných populárních webových frameworků umožnil vhled do jejich vnitřních procesů. Všechny poznatky budou velmi důležité v následujícím návrhu nového a moderního frameworku v nízkourovňovém programovacím jazyce C++ s důrazem na rychlost, přehlednost a bezpečnost.

Kapitola 4

Návrh řešení

Tato kapitola se věnuje návrhu webového frameworku, který bude vycházet z analýzy populárních projektů zpracované v předešlé části této práce. Cílem návrhu je nejen reflektovat zjištěné poznatky o obecných strukturách a komponentách, ale i nedostatky, jež vyplynuly podrobným zkoumáním jednotlivých částí jejich dokumentace a zdrojových kódu.

4.1 Programovací jazyk

Historie programovacích jazyků používaných pro vývoj webových aplikací je stejně dlouhá jako historie samotného internetu. V této evoluci se stal nejúspěšnější jazyk PHP, který je doposud jedním z nejpoužívanějších pro vývoj v tomto odvětví. Komunita tohoto jazyka je velmi aktivní a vydává nové verze, které se snaží vyřešit nedostatky jako jsou výkon, paměťová náročnost a funkcionalita. I přes všechnu snahu jsou tyto vlastnosti stále ve fázi vylepšování a prozatím nedosáhly svého maxima. Na tyto nedostatky narážela například společnost Meta (dříve Facebook), jež provozuje několik sociálních sítí. V počátku používala právě jazyk PHP pro obsluhování uživatelů, ale s rostoucí uživatelskou základnou byla nucena přejít k alternativním řešením, mezi které patří jazyk C++ [22].

V průběhu let se ve světě technologií začaly objevovat nové jazyky podporující objektově orientované programování, snadnou přenositelnost, jednoduchost atp. Avšak tyto vlastnosti jsou zpravidla nepřímo úměrné efektivitě zdrojového kódu a následnému využití systémových zdrojů. Mezi jejich zástupce patří například dnes populární Python, Java nebo ASP.NET.

Nicméně žádný z výše uvedených jazyků zatím významně nepřekonal jazyk C++ v rychlosti, jelikož je samotná aplikace výsledkem kompilovaného a optimalizovaného kódu pro daný hardware. Dalším významným důvodem jeho využití je široká paleta funkcí, jež se rozšiřuje s novými standardy [23], a také to, že se lze za pomoci vhodných knihoven vyvarovat manuální správě paměti, což bylo označováno jako jedna z hlavních nevýhod. Právě díky

pokroku tohoto jazyka lze říci, že je vhodný nejen pro vývoj systémových aplikací, ale i těch internetových.

Na druhou stranu z jazyka C++ také plynou určité nevýhody, které se projeví především v počátcích jeho užívání. V první řadě je nutné si uvědomit, že se jedná o silně typový jazyk, ve kterém má vývojář tzv. poslední slovo. Také je třeba uzpůsobit myšlení při implementacích projektů tak, aby kód byl skutečně nenáročný na dostupné zdroje, k čemuž je nezbytné znát základní principy fungování procesoru a paměti. Na závěr je třeba seznámt se s řadou technologií, jako je například `cmake`, a pointerovou aritmetikou, která je do určité míry unikátní pro jazyky C a C++.

4.2 Cílová skupina

Implementace bude svým rozsahem, dokumentací a funkcionalitou navržena tak, aby se neorientovala pouze na určitou skupinu vývojářů. Webový framework bude dostupný pro každého, kdo bude chtít těžit z vlastností a výhod, které toto řešení bude nabízet svým uživatelům.

4.3 Licence

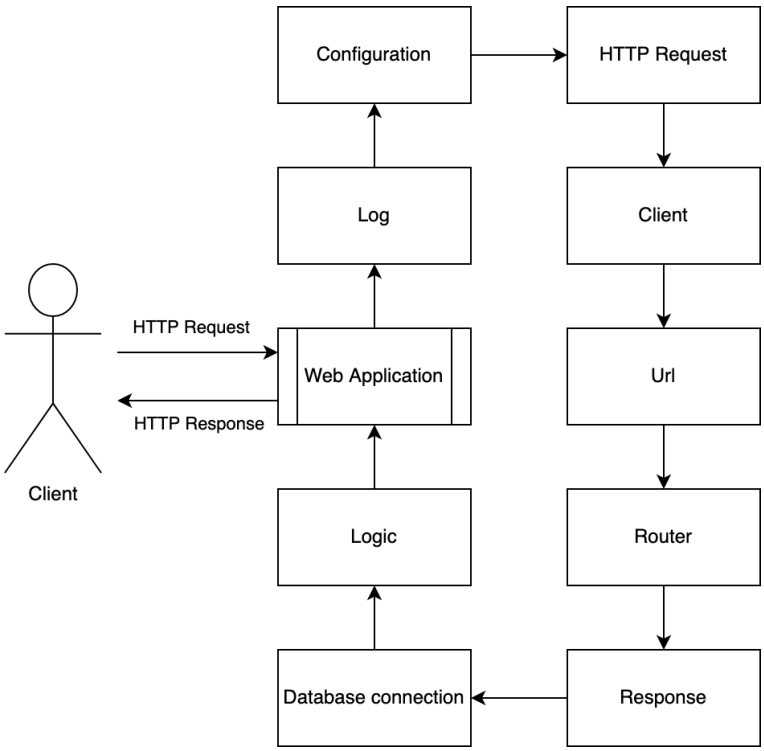
Studium licencí speciálně vytvořených pro daný software je velmi náročné, a proto vývojáři často hledí, aby produkty, které chtějí využít jako součást vlastních řešení, byly publikovány pod známými a přívětivými licencemi. Do této kategorie lze jednoznačně zařadit tzv. svobodné licence, které umožňují relativně neomezeně nakládat s autorskoprávně chráněnými díly za splnění minimálních podmínek [24].

Jak již vyplynulo z analýzy, Nette, Laravel i Django jsou šířeny pod různými svobodnými licencemi především z výše uvedených důvodů, a proto bude navržený webový framework také šířen pod svobodnou licencí GNU General Public License v druhé a zároveň i v novější verzi, jež detailně definují podmínky použití [25], aby vyšel maximálně vstříc svým uživatelům a jejich požadavkům.

4.4 Životní cyklus

Životní cyklus hrál důležitou roli při popisu fungování jednotlivých webových frameworků a následné identifikaci obecné struktury, základních tříd a funkcionalit. Zároveň jeho detailní popis umožnil pochopení vnitřních procesů, které se vykonávají na straně serveru při každém zpracování HTTP požadavku uživatele.

Poznatky získané podrobnou analýzou populárních řešení vedly k návrhu takového životního cyklu (viz obrázek 4.1), který logicky odděluje jednotlivé kroky webového frameworku do elementárních struktur tak, aby plnily konkrétní účel. Takovéto dělení je stěžejní pro implementaci, jelikož určuje konkrétní strukturu celého projektu a udává směr, kterým se bude v budoucnu ubírat.



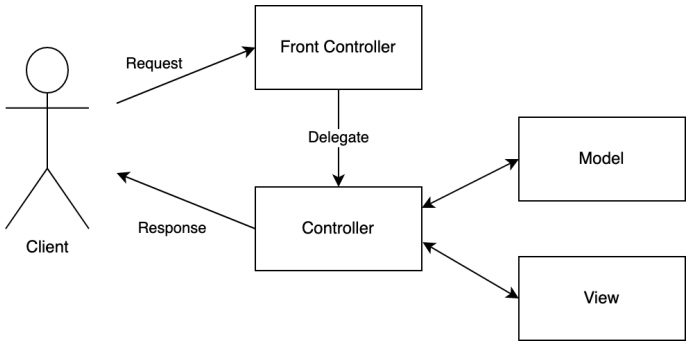
Obrázek 4.1: Návrh životního cyklu

Výše uvedený návrh vychází z obecné struktury popsané v závěru analýzy již existujících řešení (viz obrázek 3.23). Byly především určeny základní komponenty, které jsou například v dokumentaci Laravel pojmenovány jako *poskytovatelé služeb*. Jejich výčet je vzhledem k principu fungování jazyka C++ statický a nelze jej nadále rozšiřovat. V případě dalších individuálně žádoucích komponent je dostupný prostor v kontroléru a modelu, který je pod absolutní správou vývojáře.

4.5 Architektura

V návrhu životního cyklu (viz obrázek 4.1) je závěrečný krok označen jako *logic*. Pod tímto pojmenováním se skrývá návrhový vzor, jehož funkce byla popsána v sekci věnující se této problematice. Výše uvedené webové frameworky se svojí architekturou částěčně liší, i přesto, že jsou určeny ke stejnému účelu. V případě této implementace bude použit již zmíněný *Model-View-Controller*

(viz obrázek 4.2) doplněný o tzv. *front controller*, což s sebou nese několik strukturálních výhod.



Obrázek 4.2: MVC architektura obsahující front controller

Máme-li aplikaci skládající se z více kontrolérů, s velkou pravděpodobností se budou některé jejich jednotlivé části opakovat. Takovým příkladem opakovaných činností je například připojení do databáze, načtení konfiguračního souboru či vytvoření objektu pro logování, které se budou volat při zpracovávání každého HTTP požadavku. Uspořádání společných částí do jedné sekce zdrojového kódu zamezuje jejímu duplikování a tím jej zároveň zpřehledňuje. Tato sekce se nazývá *front controller* [26]. Další výhodou tohoto konceptu je snazší migrace k protokolu FastCGI, který přímo vyžaduje určitý prostor před samotným přesunem do konkrétní logiky (viz obrázek 4.3).

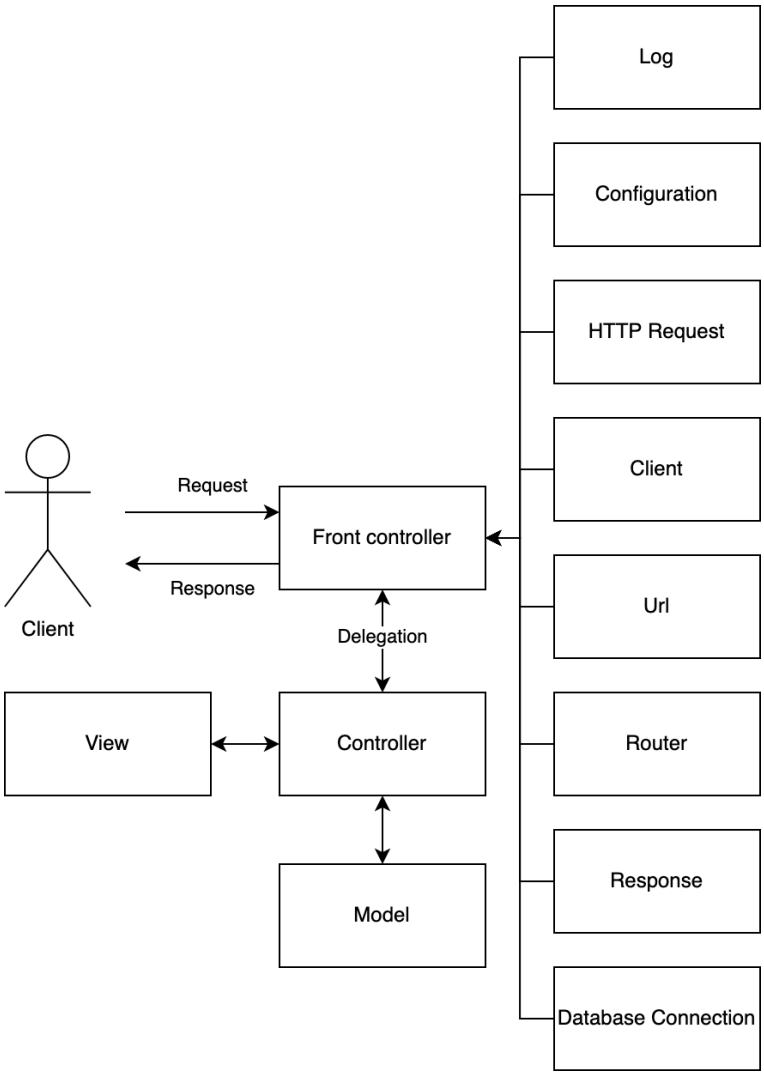
```
#include <iostream>
#include "fcgio.h"
// main function as front controller
int main(void) {
    FCGX_Request request;
    FCGX_Init();
    FCGX_InitRequest(&request, 0, 0);
    // connect to database + read configuration
    // wait for requests
    while (FCGX_Accept_r(&request) == 0) {
        // logic
    }
   return 0;
}
```

Obrázek 4.3: Ukázka webové aplikace za užití protokolu FastCGI

Protokol FastCGI oproti CGI, jež při každém dotazu spouští skript či aplikaci jako samotný proces, posílá data do souběžně běžící aplikace, která čeká na vstup. Tento přístup uspoří nemalé množství systémových zdrojů, jež jsou nutné pro spuštění nové aplikace. Zároveň se duplicitní činnosti provedou pouze jednou a poté jsou jejich výsledky dostupné pro následující dotazy.

4.6 Technický návrh

Spojením rozšířené MVC architektury (viz obrázek 4.2) a životního cyklu (viz obrázek 4.1) lze sestavit konkrétní technický návrh celého webového frameworku, který bude reflektovat postavení jednotlivých komponent, předsunutého kontroléru a průběh HTTP požadavku ve výsledné webové aplikaci (viz obrázek 4.4).

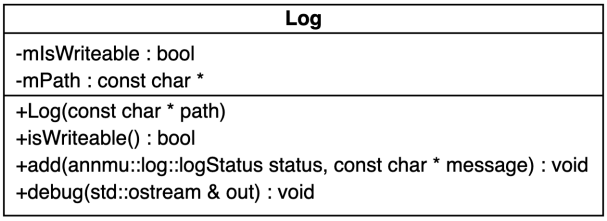


Obrázek 4.4: Technický návrh aplikace

Společné části význačné pro každý příchozí dotaz se provedou v předsunutém kontroléru. Konkrétní části webové aplikace - implementované samotnými uživateli - budou odděleny do samostatných pohledů, modelů a kontrolérů, jež budou mít k dispozici veškeré objekty vytvořené v předešlém kroku a případně další podpůrné komponenty obsažené ve webovém frameworku.

4.6.1 Log

První krok, který by webový framewrok měl provést, je vytvoření objektu pro zaznamenávání událostí. V konstruktoru přijme cestu k souboru, do něhož bude zaznamenávat (viz obrázek 4.5). Současně zkontroluje, zda disponuje oprávněními do něj zapisovat. Na tuto významnou skutečnost se bude možné kdykoliv dotázat užitím metody `isWriteable()`. Podstatnou součástí implementace bude také aktivní využívání funkce `fflush()`, jež slouží k vyprazdňování vyrovnávací paměti [27], při každém pokusu zápisu do souboru. V případě neočekávaného selhání aplikace by bez jejího použití k této činnosti nemuselo dojít a informace o stavu by byly nenávratně ztraceny, což by mělo za následek ztížené odhalování chyb.



Obrázek 4.5: Diagram Log třídy

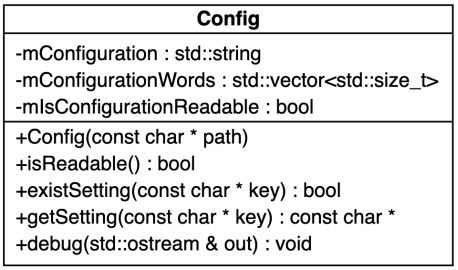
4.6.2 Konfigurace

V jazycích PHP či Python, ve kterých jsou analyzované projekty implementovány, je možné konfiguraci zapisovat přímo do zdrojového kódu, jelikož se dále nekompiluje a tím pádem se změny propíšou okamžitě bez nutnosti dalších zásahů [28, 29]. V případě jazyka C++ je nezbytné, aby veškerá nastavení, která se musí projevit v reálném čase při jejich změně, byla zapsána do odděleného textového souboru. Forma zápisu v implementaci by měla být intuitivní a přímočará. Na každém řádku se bude nacházet dvojice klíč-hodnota. Bude-li řádek prázdný nebo začínat mřížkou, dojde k jeho přeskočení. Příkladem konfigurace mohou být například přihlašovací údaje do databáze či informace o ladícím módu (viz obrázek 4.6).

```
# prefix for DB tables and directories
prefix = site_

# the database information
dbAddr = tcp://localhost:3306
dbUser = admin
dbPass = admin
dbName = annmu
```

Obrázek 4.6: Ukázka konfiguračního souboru

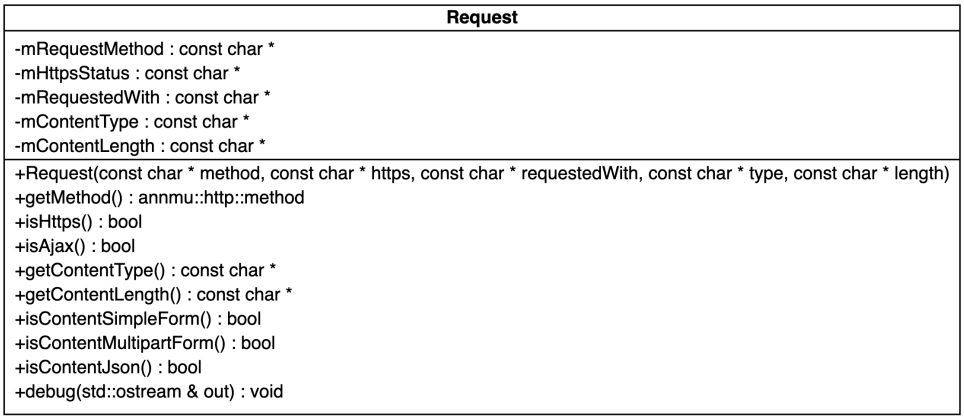
Přímé ovlivňování zdrojového kódu obnáší určitá rizika, jelikož nesprávný zápis způsobený neznalostí či nepozorností může vyústit v poškození aplikace nebo změně jejího chování. Vytvořením třídy načítající konfiguraci z externího zdroje lze předejít mnoha problémům za pomoci řádného ošetření jednotlivých vstupů. Implementovaná třída zpracovávající konfiguraci nabídne základní paletu metod, které poskytnou nejen jednotlivé informace o datech, ale i potvrzení jejich správného načtení (viz obrázek 4.7). Samotnou ochranu souboru před vnějším čtením lze zajistit například na úrovni serveru [30].



Obrázek 4.7: Diagram Config třídy

4.6.3 HTTP Požadavek

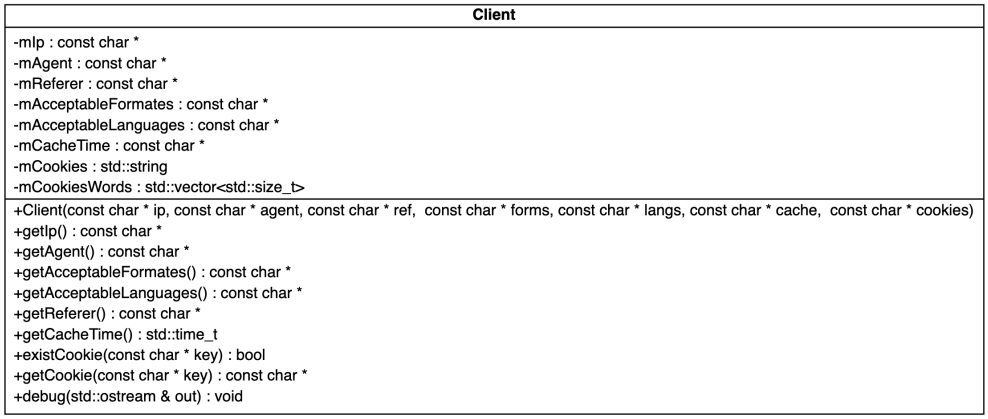
Třída zpracovávající HTTP požadavek uchová základní informace o dotazu, mezi které patří například metoda požadavku nebo použitý protokol pro komunikaci, a poskytne náležité metody, které jej nabídne zpracované ve vhodném formátu (viz obrázek 4.8). Na rozdíl od analyzovaných řešení byly pro lepší přehlednost úplně vyčleněny do samostatných tříd informace o URL adrese a samotném klientovi. Kromě běžně používaných metod, tedy GET a POST, existuje několik dalších pro komunikaci se serverem, které mají speciální účely [31], a je nutné umět se s nimi vypořádat. Tento problém lze v implementaci řešit za účelem optimalizace již v předsunutém kontroléru a poté regulérně ve směrovači. V případě použití frameworku neznámé metody je možné navrátit chybový status `400 BAD REQUEST`.



Obrázek 4.8: Diagram HTTP Request třídy

4.6.4 Klient

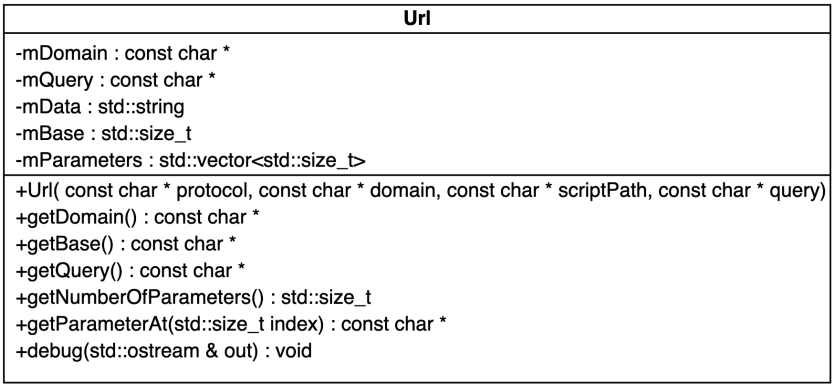
Jedním z důvodů separování informací o klientovi od třídy zpracovávající HTTP požadavek je především snadná škálovatelnost metod zajišťujících zpracovávání informací ze získaných dat v budoucích úpravách webového frameworku při zachování systematického členění. Zakladní metody navržené třídy (viz obrázek 4.9) nabízí informace o uživateli jako celek, avšak vzhledem k vývoji evropské legislativy a zásad ochrany osobních údajů [32] bude pravděpodobně vhodné některé statistiky o přijatých dotazech provádět na úrovni serveru. Mezi takové statistické údaje lze jednoznačně zařadit například název webového prohlížeče, operační systém či typ zařízení. Navržení samostatné třídy je tedy vhodným krokem k přípravě na možné budoucí překážky, které nemusí pocházet pouze z technického prostředí.



Obrázek 4.9: Diagram Client třídy

4.6.5 URL

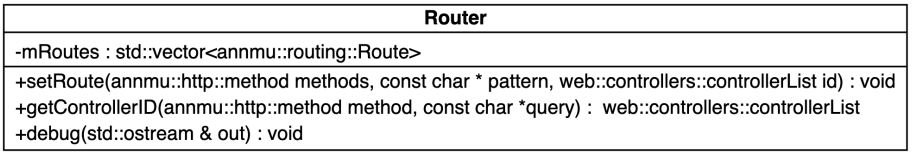
Obdobně jako v analyzovaném webovém frameworku Nette by měly být informace o dotazované URL adrese pro lepší přehlednost vyčleněny do samostatné třídy. Zároveň by zde mělo být obsaženo několik rozšířených funkcionalit (viz obrázek 4.10), které by umožnily získání aktuální základní URL adresy aplikace k vytváření absolutních cest a podporu tzv. hezkých URL, které jsou pro uživatele snadno čitelné, zapamatovatelné a z hlediska SEO nezbytné [33]. Vyznačují se používáním znaku lomítka jako oddělovače jednotlivých parametrů. Příkladem takové adresy je například `example.com/articles/cooking/how-to-cook-fish`.



Obrázek 4.10: Diagram URL třídy

4.6.6 Směrovač

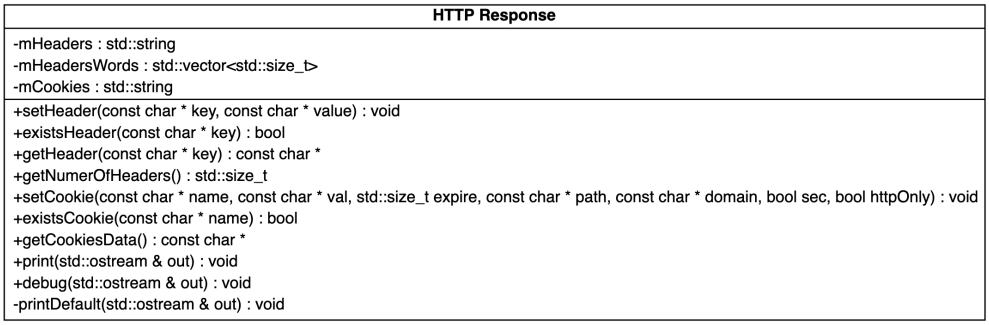
Úkolem směrovače je určit, jaký další kód se bude vykonávat na základě poskytnutých informací. Třída by měla implementovat jednoduchou funkcionalitu na vytváření nových cest a následné dotazování se na ně (viz obrázek 4.11). Vnitřní uspořádání dat by se mělo provádět do vlastní datové struktury, která umožní případné ladění při vývoji jednotlivých částí webové aplikace.



Obrázek 4.11: Diagram Router třídy

4.6.7 HTTP odpověď

Ke každému požadavku je nutné vytvořit odpověď, která bude obsahovat mimo jiné hlavičku s metadaty, jež mohou ovlivnit nejen vykreslování výsledné stránky, ale zásadním způsobem i její bezpečnost. Do této hlavičky se také zapisují nové soubory cookies, které se mají v prohlížeči uživatele vytvořit. V případě, že hlavička zůstane nedotčena, měla by se vypsat výchozí metadata, jež budou obsahovat maximální množství doprovodných informací poskytující vysokou míru ochrany před potencionálními útoky (viz obrázek 4.12).



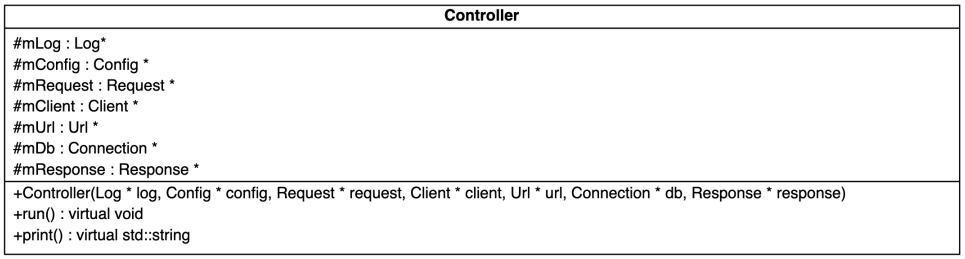
Obrázek 4.12: Diagram HTTP Response třídy

4.6.8 Databáze

Na rozdíl od skriptovacího jazyka PHP v C++ neexistuje standardní knihovna, jež by umožňovala jednoduché připojení k databázi. Avšak existuje mnoho veřejně dostupných alternativ pod svobodnými licencemi, které tento nedostatek řeší. Vzhledem k použití MySQL se nabízí varianta *MySQL connector library for C and C++ applications* poskytující snadnou instalaci a použití s velmi podobnou syntaxí a názvoslovím, které je v prostředí databázových systémů a programovacích jazyků ustálené [34].

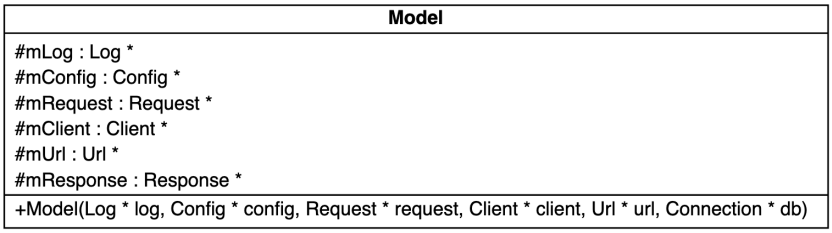
4.6.9 Model-View-Controller

Každá část webové aplikace by měla být pro lepší přehlednost odvozena od vzoru MVC a v případě implementace by měly být děděny její znaky. Model i kontrolér přijmou v konstruktoru ukazatele na objekty vytvořené v předešlých krocích (viz obrázky 4.13 a 4.14). Zvolení ukazatelů namísto referencí je především proto, že mohou být nulové, tedy neukazovat nikam. To je vhodné v případě, že nebudeme chtít využívat některou z předešlých částí a jednoduše ji v předsunutém kontroléru vynulujeme. Konkrétní logika se bude unifikovaně zapisovat do metody `run()`, která bude frameworkem automaticky zavolána. Následně metoda `print()` vrátí řetězec, který se má vypsat na výstup.

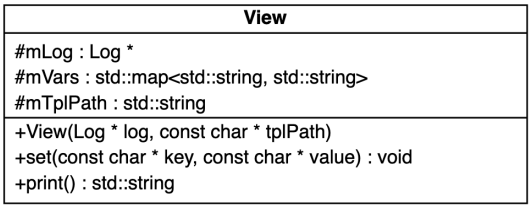


Obrázek 4.13: Diagram MVC Controller třídy

Třída `View` bude založena na textových šablonách, ve kterých bude hledat značky, jež následně přepíše za data přijatá metodou `set()` (viz obrázek 4.15) a zavoláním metody `print()` vrátí jejich úplné znění. Více pokročilých funkcionalit, mezi které patří například forcykly, není nezbytné v základní verzi implementovat, jelikož je lze ekvivalentně vytvořit přímo v navržené třídě.



Obrázek 4.14: Diagram MVC Model třídy



Obrázek 4.15: Diagram MVC View třídy

Kapitola 5

Implementace

V této kapitole se čtenář seznámím se strukturou navrženého webového frameworku, který pro lepší přehlednost dostal název ANNMu (s výslovností [anmu]). Taktéž bude popsán systém dokumentace zdrojového kódu, průběh implementace a komponenty, které byly vytvořeny nad rámec technického návrhu z předešlé části této práce.

5.1 Struktura

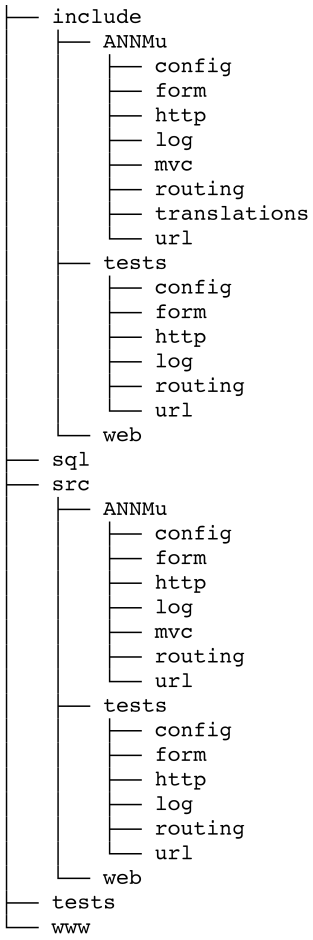
Základní struktura projektu (viz obrázek 5.2) z velké části reflektuje technický návrh (viz obrázek 4.4). Hlavními podsložkami kořenového adresáře jsou `include` a `src`, které se následně dělí na další podsložky dle části logiky, kterou implementují. Zároveň je toto uspořádání respektováno ve zdrojovém kódu, ve kterém jsou stejným způsobem vytvořeny jmenné prostory pro snazší orientaci ve webovém frameworku. Budeme-li tedy například hledat zdrojový kód třídy Config (viz ukázka 5.1), tak jeho adresářová cesta bude `src/ANNMu/config/config.cpp`, tedy identicky s jmennými prostory.

```
/*
 *
 * Configuration
 *
 */

// create object
annmu::config::Config config("web.config");

// check configiguration file
if(!config.isReadable()) {
    log.add(annmu::log::LOG_ERROR, "Configuration is not readable!");
    std::cout << "Status:" << annmu::http::STATUS_500_INTERNAL_SERVER_ERROR << "\n\n";
}
```

Obrázek 5.1: Ukázka jmenných prostorů

V kořenovém adresáři se také náchází složka `www`, do které se výsledná aplikace zkompiluje. V této složce jsou připravené všechny nezbytné soubory pro správný běh aplikace a je uzpůsobena k tomu, aby ji bylo možné zkopírovat do příslušného adresáře webového serveru jako již hotový a plně funkční projekt.

Kromě adresáře `sql`, jež obsahuje pomocné databázové tabulky k některým rozšiřujícím komponentám, je zde připravena složka `tests` pro samotné testování správné funkčnosti webového frameworku na různých zařízeních a po úpravách zdrojového kódu.



Obrázek 5.2: Struktura projektu

5.2 Dokumentace

Každý hlavičkový soubor v adresáři `icnlude` obsahuje základní informace o licenci, autorovi a projektu, jehož je součástí. Všechny funkce a metody tříd jsou řádně zdokumentovány, tedy obsahují popis jejich účelu, vstupu a výstupu. Pro zápis byl použit syntax *Javadocu* (viz ukázka 5.3) a to především proto, že mu rozumí velké množství editorů a komentáře v jazyku C++ jsou identické jako v jazyku Java.

```
/**
 *
 * @brief The static method decodes the given array.
 *
 * @param str It's the pointer to the array that contains
 * the encoded string.
 *
 * @return The static method returns the decoded array.
 *
 */

static std::string decode(const char * str);
```

Obrázek 5.3: Ukázka dokumentace funkce napsané v Javadocu

5.3 Průběh implementace

Implementace se řídila diagramy z technického návrhu a nebylo nutné provést strukturální změny. Pouze byly přidány utility, které usnadňovaly vývoj některých komponent. Jednalo se především o práci s URL adresou a formuláři. Kompletní zdrojový kód byl napsán ve standardu z roku 2011. Tato verze nabízí jistotu stability jazyka a také je doporučena v dokumentaci použité MySQL knihovny v případě vlastní kompilace[1].

Žádná z jádrových funkcionalit není závislá na databázi a tedy připojení do ní není pro správnou funkčnost webového frameworku nijak podmíněno. Veškerý kód byl navržen tak, aby prováděl minimum činností a byl volán pouze v situacích, kde je to nezbytně nutné. Současně pro ověření správného chování byly vytvářeny jednotkové testy k většině metod a funkcím. V závěru návrhů většiny tříd je uvedena metoda `debug()`, jež byla implementována tak, aby zpracovávala dostupné informace a vypisovala je do streamu z parametru ve formátu přívětivém pro konzole webových prohlížečů. Jejich volání lze ovlivnit v konfiguračním souboru aplikace, jelikož mohou vypisovat velmi

[1] viz dokumentace MySQL: https://dev.mysql.com/doc/connector-cpp/8.0/en/connector-cpp-apps-general-considerations.html

citlivé údaje. Jedná se o majoritní ladící nástroj, který umožňuje procházet příchozí informace a zhodnotit, jak s nimi webový framework pracuje.

Kapitola 6

Ověření

Nedílnou součástí práce je ověření cílů stanovených v úvodu. Především se jedná o oblast rychlosti, tedy doba zpracování požadavku na serveru a s tím spojené využití systémových zdrojů. Dále byl kladen důraz na bezpečnost celého řešení, aby nedošlo ke kompromitaci serveru a v závěru také na použitelnost a konkurenceschopnost.

6.1 Použití

Ukázka použití vytvořeného webového frameworku na reálném příkladu je důležitá, jelikož se tím ověří možnost uplatnění v prostředí internetu. Pro ukázku byl vybrán web Pivovaru BORN, jehož cílem je sloužit jako prezentace pro veřejnost, což lze zobecnit na požadavky většiny webových stránek společností. Z technického hlediska je vhodným kandidátem, jelikož obsahuje základní funkcionality vyžadované většinou webových aplikací, mezi které patří menu, slidery, aktuality, kontaktní formuláře atd.

Vstupem byla statická HTML šablona (viz ukázka 6.1), která definovala grafickou podobu webové stránky. Výstupem měla být dynamická webová aplikace, jež bude propojena s databází, ze které získá aktuální obsah, a zároveň bude reagovat na změny dat v reálném čase (při přenačtení stránky).

Výsledný zdrojový kód webové aplikace je součástí této práce jako externí příloha. Navržený webový framework přímo vede ke struktuře projektu a zároveň řeší základní problematiku, například jakou část kódu pro danou url adresu spustit či jak získat data o HTTP požadavku atd. Bylo tedy nutné pouze vytvořit konkrétní logiku. Díky architektuře MVC bylo jasně stanoveno, jakým způsobem ji implementovat. Doba programování se pohybovala v řádu nižších desítek hodin a odpovídá přibližnému odhadu práce v běžném PHP frameworku.

Obrázek 6.1: Ukázka stránky Pivovaru BORN

Výsledný čas vykreslování jedné stránky webové aplikace se pohybuje okolo 14 milisekund. Odečteme-li čas vytváření připojení do databáze, zjistíme, že aplikace i s načtením webového frameworku potřebuje přibližně 4 milisekundy. Téměř všechny potřebné funkcionality nezbytné k vytvoření webové aplikace byly k dispozici, avšak v budoucích updatech webového framewroku by bylo vhodné doplnit pomocné funkce pro lepší práci s HTML tagy a pokročilými formami HTML formulářů.

V závěru je nutno podotknout, že existuje mnoho dalších příkladů využití. Vzhledem k poskytnutému výkonu a relativně nízkým paměťovým nárokům je webový framework vhodný například pro vytváření API rozhraní nebo sběr dat za pomocí HTTP protokolu.

6.2 Rychlost

Provést validní měření rychlosti webových frameworků je poměrně obtížné, jelikož zkoumané projekty neprovádí identickou činnost. Jednotlivé testy by taktéž měly být omezené pouze na funkcionalitu bez jakýchkoliv komponent vytvořených samotnými uživateli a z měření by měla být odečtena doba vytváření připojení do databáze, jelikož zpravidla závisí na databázovém serveru. Taktéž v době měření by neměly být spuštěny zatěžující procesy na pozadí a případné cacheování. Z hlediska paměti by mělo být měřeno množství jejího využití v průběhu chodu aplikace. Jde tedy o paměť nutnou pro vytváření objektů, proměnných, načítání souborů apod.

Měření (viz tabulka 6.1) jsou průměrem 10 nezávislých měření, ve kterých se generovala základní prázdná HTML stránka, a proběhly na zařízení Apple MacBook Pro 2017 128 GB za použití webového serveru Apache 2.4.46, novější verze PHP 8.0 a Pythonu 3.9.6.

Framework	Čas	Paměť
Nette 3.1	44.036ms	3.8MB
Laravel 8.0	155.43ms	13.9MB
Django 4.0	5.319ms	196.6kB
ANNMu	1.352ms	326kB

Tabulka 6.1: Výsledky měření rychlosti webových frameworků

Výsledky odpovídají přibližnému očekávání. Instalace Laravel je poměrně rozsáhlá a využívá několik externích komponent. Stažení a vytvoření nového projektu je nutné provést přes composer, což je nástroj určený pro správu knihoven a jiných zdrojů. Při spuštění aplikace se mnoho těchto zdrojů načítá, čemuž odpovídá výsledek měření. Oproti projektu Laravel je Nette méně náročné. Instalaci lze provést stažením zazipovaného projektu, který obsahuje i debugovací nástroj Tracy a šablonovací systém Latte. Aplikace vytvořená v Nette načítá méně komponent a její konstrukce je celkově minimalistická, a proto jsou výsledky měření poměrně přívětivé. Django používá zmíněné wsgi, který lze svým principem přirovnat k FastCGI protokolu. Díky paralelnímu běhu s webovým serverem jsou již některé komponenty načtené. Z tohoto důvodu je výsledek měření velmi dobrý. Navíc, při opakovaném požadavku dojde ke cacheování a výsledný čas se může pohybovat okolo jedné milisekundy. Implementovaný webový framework je díky optimalizaci kompileru velmi rychlý. Při použití FastCGI protokolu je možné některé činnosti zredukovat a tím snížit paměťovou i časovou náročnost pro zpracování požadavku obdobně jako v případě Django.

6.3 Bezpečnost

Z definice kyberbezpečnosti (2.5) plyne několik zásad pro její dosažení. V rámci programátorské části bylo možné implementovat pouze ty principy, které se týkají přímo zdrojového kódu.

Hlavním pilířem zajištění bezpečnosti byly jednotkové testy. Celkem jich bylo vytvořeno okolo dvou set a zaměřovaly se na testování chování jednotlivých částí webového frameworku při validním, nevalidním, potencionálně nebezpečným i prázdným vstupem. Zároveň je možné pomocí nich ověřit správnou funkčnost na různých zařízeních, jelikož kód nemusí fungovat na všech operačních systémech a hardwaru, i když po celou dobu implementace byl kladen důraz na používání pouze standardních knihoven.

Dále byl důležitý výběr knihovny, jež umožní komunikaci s databázovým serverem. *C++ MySQL Connector* pro svůj běh používá například knihovny SSL a Crypto. Zároveň implementuje tzv. *prepared statements*, které se snaží předejít SQL injection, což je způsob útoku používaný především k odcizení či poškození dat uložených v databázi.

V neposlední řadě byly implementovány prvky ovlivňující bezpečnost nejen webové aplikace, ale i samotného uživatele. Ve formulářích lze použít tokeny pro ochranu před Cross-site request forgery útokem, pro práci s daty na straně serveru je možné kódovat data do bezpečné URL formy, lze také použít výchozí hlavičku HTTP odpovědi obsahující restriktivní nastavení apod.

Výše uvedené body jsou hlavními mediátory, které lze začlenit pro dosažení bezpečeného zdrojového kódu. Další majoritní i minoritní prvky, mezi které patří například ukázky použití či dokumentace, byly také v průběhu implementace brány na zřetel a částečně uplatněny.

6.4 Rozsah

Rozsah webového frameworku má mnohdy vliv nejen na výkon výsledné aplikace, ale i na její funkcionalitu. V kapitole věnující se analýze jednotlivých projektů bylo několikrát uvedeno, že byly pro dosažení funkčnosti použity komponenty z jiných řešení. Navržený webový framework obsahuje několik tisíc řádků zdrojového kódu implementujících základní sadu nástrojů, která umožňuje vytvořit webovou aplikaci bez nutnosti instalace dalších komponent. Avšak v případě potřeby rozsáhlejších funkcionalit je možné využít externích knihoven a zdrojových kódu obdobně, jak to dělají i jiné webové frameworky.

Kapitola 7

Závěr

Cílem práce bylo vytvořit rychlý a bezpečný webový framework napsaný v systémovém jazyce C++, který je schopen efektivně pracovat s dostupnou pamětí a jinými systémovými zdroji. Motivací vzniku takového projektu byla především autorova zkušenost s vývoji webových aplikací v tradičních programovacích jazycích a minimální dostupnost obdobných řešení, jelikož velké firmy věnující se této problematice, z důvodu redukce nutné infrastruktury způsobené používáním neefektivních technologií, nejsou příliš ochotné publikovat výsledky své práce, aby zamezily šíření svého know-how a rozvoji své konkurence.

Ze všeho nejdříve byla provedena podrobná analýza (viz kapitola 3) vybraných řešení - Nette, Laravel a Django - jež jsou obklopena početnou komunitou a těší se poměrně velké popularitě. Zcela jistě na poli internetu existuje mnoho dalších webových frameworků, které by si zasloužily být součástí analýzy, ale cílem bylo především nalézt jejich obecnou strukturu, tedy jejich uspořádání, vnitřní členění, nabídku funkcionalit či vnitřní procesy, které se provádí automaticky na pozadí. Nalezení odpovědí na všechny dílčí části byl poměrně náročný proces a to především z důvodu různorodosti použitých programovacích jazyků, neúplných dokumentací, nedokonalému uspořádání zdrojových kódů a nutné znalosti nejen principů webových stránek, ale i principů bezpečnosti a počítačových sítí. Výstupem provedené analýzy jsou životní cykly jednotlivých webových frameworků, které popisují jejich princip fungování a vnitřní procesy od spuštění aplikace až do jejího úplného konce.

V následném návrhu (viz kapitola 4) byl vytvořen životní cyklus webového frameworku a popsána MVC architektura, která má za cíl usnadnit vývoj dalších komponent a ustálit způsob jejich vytváření. Součástí popisu životního cyklu byly také diagramy jednotlivých tříd, které již dávaly budoucí implementaci konkrétní rysy. Připojení a práce s databází je velmi podstatná v prostředí internetu. Zároveň v jazyce C++ neexistuje standardní knihovna pro práci s ní, a proto bylo nutné před zahájením programovací části vybrat externí řešení, které bude poskytovat náležitou funkcionalitu a bezpečnost.

Díky podrobnému návrhu nebylo nutné v implementaci provádět významné strukturální změny. Hlavní snahou bylo zamezit duplikování dat a provádění některých činností, dokud to nebylo nezbytné. Tohoto cíle bylo dosaženo zvláště prací s konstantními ukazateli a vhodnou realizací metod navržených tříd.

Řešení bylo podrobeno praktickému použití, ve kterém bylo cílem převést HTML šablonu do dynamického stavu. V rámci ověření také došlo ke srovnání jednotlivých webových frameworků, a to v disciplíně náročnosti na systémové zdroje. Výsledky měření zřetelně prezentují rozdíly v jednotlivých technologiích (viz kapitola 6). Na závěr byl popsán princip dosažení bezpečnosti v souladu s definicí z kapitoly věnující se základním pojmům.

Cílů vymezených na počátku práce bylo uspokojivě dosaženo, což bylo ukázáno na praktické ukázce a porovnaní s jinými webovými frameworky. Implementace obsahuje základní komponenty pro vytváření různorodých webových aplikací, avšak je zde stále prostor pro pokročilejší funkcionality, jež by mohly v některých případech usnadnit vývoj projektů.

Příloha A

Literatura

[1] Nette Foundation. *Nette 3.1 Documentation*. 2021.

[2] Laravel LLC. The php framework for web artisans. `https://laravel.com/`, 2011-2021.

[3] S. Dauzon, A. Bendoraitis, and A. Ravindran. *Django: Web Development with Python*, page 5. Packt Publishing, 2016.

[4] M.Milić V. Stanojević, S. Vlajić and M. Ognjanović. Guidelines for framework development process. In *2011 7th Central and Eastern European Software Engineering Conference (CEE-SECR)*, page 1, 2011.

[5] J. Pater. Modern web application frameworks. Master's thesis, Fakulta informatiky Masarykovy univerzity, 2015.

[6] C. Bates. *Web programming: building Internet applications, 2nd*, page 284. Wiley, Chichester, 2002.

[7] D. Robinson and K. A. L. Coar. The common gateway interface (cgi) version 1.1. RFC 3875, RFC Editor, October 2004.

[8] D. Schatz, R. Bashroush, and J. Wall. Towards a more representative definition of cyber security. *J. Digit. Forensics Secur. Law*, 12:53–74, 2017.

[9] Django Software Foundation. The web framework for perfectionists with deadlines. `https://www.djangoproject.com/`, 2005-2021.

[10] N. George. *Beginning Django CMS*, page 1. Apress, 2015.

[11] Laravel LLC. *Laravel 8.x documentation*. 2021.

[12] Django Software Foundation. *Django 3.2 documentation*. 2021.

[13] N. Cooper and K. Gee. *Build Your Own Website: A Comic Guide to HTML, CSS, and WordPress*. No Starch Press, 1st edition, 2014.

[14] B. Totty, D. Gourley, M. Sayer, A. Aggarwal, and S. Reddy. *HTTP: The Definitive Guide*, pages 46–47. O'Reilly and Associates, Inc., 2002.

[15] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform resource identifier (uri): Generic syntax, 2005.

[16] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*, page 3. Addison-Wesley Longman Publishing Co., Inc., 1995.

[17] W. J. Gilmore. *Easy Laravel 5*, page 52. Learnpub, 2018.

[18] A. Ravindran. *Django Design Patterns and Best Practices*, page 15. Packt Publishing, 2015.

[19] D. Voorhees. *Guide to Efficient Software Design: An MVC Approach to Concepts, Structures, and Models*, page 176. 2020.

[20] J. Chan. *PHP: Learn PHP in One Day and Learn It Well. PHP for Beginners with Hands-on Project.* LCF Publishing, 2020.

[21] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Rfc2616: Hypertext transfer protocol – http/1.1, 1999.

[22] H. Zhao. Hiphop for php: Move fast. `https://developers.facebook.com/blog/post/2010/02/02/hiphop-for-php--move-fast/`, 2012.

[23] B. Stroustrup. *The C++ Programming Language.* Addison-Wesley Professional, 2013.

[24] M. Bernelin. The compatibility of open/free licences: a legal imbroglio. *International Journal of Law and Information Technology*, 28(2):93–111, 08 2020.

[25] Free Software Foundation. Gnu general public license version 3, 2007.

[26] M. Fowler. *Patterns of Enterprise Application Architecture.* Addison-Wesley Educational, 2002.

[27] B. W. Kernighan and D. M. Ritchie. *C Programming Language.* Prentice Hall, 2 edition, 1988.

[28] F. M. Kromann. *Beginning PHP and MySQL.* APRESS, 5 edition, 2018.

[29] J. P. Mueller. *Beginning programming with python for dummies.* John Wiley & Sons, 2 edition, 2018.

[30] T. Steidler-Dennison. *Run your own web server using Linux and Apache.* SitePoint, 2005.

[31] B. Pollard. *HTTP/2 in Action.* Manning Publications, 2019.

[32] P. Voigt and A. von dem Bussche. *The EU general data protection regulation (GDPR).* Springer International Publishing, 2017.

[33] F. Caldarelli. *Yii2 By Example.* Packt Publishing, 2015.

[34] Oracle and its affiliates. *MySQL Connector/C++ Documentation.* 2015-2020.

Příloha B

Technická dokumentace

B.1 Závislosti

Webový framework je primárně vytvořený pro webový server Apache a MySQL, jelikož se jedná o běžně používané technologie. Pro správný běh výsledné webové aplikace je nutné pouze nainstalovat *C++ MySQL connector* z oficiální webové stránky[1] a v souboru `CMakeLists.txt`, jež se nachází v kořenovém adresáři projektu, upravit cesty k hlavičkovým souborům a nainstalované knihovně.

B.2 Kompilace

Za pomocí příkazu `cmake CMakeLists.txt` se vygeneruje `Makefile`, který po zavolání vytvoří samotnou webovou aplikaci `index.cgi` v podadresáři `www` a aplikaci `tests.cgi` v podadresáři `tests`. Spuštěním aplikace s testy lze ověřit, že framework funguje tak, jak má. Složku `www` je možné zkopírovat do příslušného adresáře webového serveru a tím by měla být webová aplikace plně funkční. Může se stát, že ve výchozím nastavení serveru není dovoleno spouštět CGI skripty, to lze změnit úpravou konfiguračního souboru[2].

B.3 Tvorba komponent

Webová aplikace se zkládá z komponent. Každá komponenta se skládá ze tří částí - kontrolér, model a pohled. Tyto části jsou reprezentovány třídami, které dědí základní vlastnosti od MVC modelu implementovaného ve webovém frameworku. Nové komponenty se vytvářejí pro přehlednost do adresáře s názvem `web` (viz ukázka B.1).

[1] viz odkaz: https://dev.mysql.com/downloads/connector/cpp/8.0.html
[2] viz dokumentace Apache: https://httpd.apache.org/docs/2.4/howto/cgi.html

```
.
└── web
    ├── beer
    │   ├── controller.cpp
    │   ├── model.cpp
    │   └── view.cpp
    ├── beers
    │   ├── controller.cpp
    │   ├── model.cpp
    │   └── view.cpp
    ├── contact
    │   ├── controller.cpp
    │   ├── model.cpp
    │   └── view.cpp
    ├── event
    │   ├── controller.cpp
    │   ├── model.cpp
    │   └── view.cpp
    ├── events
    │   ├── controller.cpp
    │   ├── model.cpp
    │   └── view.cpp
    ├── homepage
    │   ├── controller.cpp
    │   ├── model.cpp
    │   └── view.cpp
    ├── restaurant
    │   ├── controller.cpp
    │   ├── model.cpp
    │   └── view.cpp
```

Obrázek B.1: Ukázka uspořádání komponent uvnitř adresáře

Pro snazší pochopení je součástí frameworku připravena ukázková komponenta `homepage`, která obsluhuje hlavní stranu webové aplikace. Při vytváření nové komponenty je jako první krok nezbytné vytvořit číselné ID pro kontrolér v hlavičkovém souboru `include/web/controllerList.hpp` (viz ukázka B.2), které bude používáno jako identifikátor komponenty v rámci směrovače.

```
namespace web {

    namespace controllers {

        enum controllerList : int {
            UNKNOWN = 0,            // do not touch
            CONTROLLER_HOMEPAGE,
            CONTROLLER_NEWS,
            CONTROLLER_CONTACT
        };


    }

}
```

Obrázek B.2: Ukázka výčtu ID kontrolérů

Po vytvoření nového ID je nutné v `src/web/routerFactory.cpp` definovat, pro jakou URL adresu a metody dotazu se má komponenta zavolat (viz ukázka B.3). Zde je nutné mít kontrolér označený číselně, jelikož jazyk C++ nepodporuje předávání ukazatelů na konstruktory tříd jako parametry funkcí či metod.

```
annmu::routing::Router RouterFactory::create() {

    annmu::routing::Router router;

    // homepage
    router.setRoute(
        annmu::http::METHOD_GET,
        "^$",
        web::controllers::CONTROLLER_HOMEPAGE
    );

    // contact
    router.setRoute(
        annmu::http::METHOD_GET_POST,
        "^kontakt$",
        web::controllers::CONTROLLER_CONTACT
    );

    // events
    router.setRoute(
        annmu::http::METHOD_GET,
        "^udalosti$",
        web::controllers::CONTROLLER_EVENTS
    );

    // event
    router.setRoute(
        annmu::http::METHOD_GET,
        "^udalost/[a-zA-Z0-9_-]+$",
        web::controllers::CONTROLLER_EVENT
    );

    return router;

}
```

Obrázek B.3: Ukázka továrny na směrovač

Dále, po implementování jednotlivých částí MVC do složky `web`, jej stačí přidat do továrny (`src/web/controllerFactory.cpp`), jež vytváří kontroléry na základě onoho číselného ID (viz ukázka B.4).

```
annmu::mvc::Controller * ControllerFactory::create(/* ... */) {

    if(id == web::controllers::CONTROLLER_HOMEPAGE) {
        return new web::homepage::Controller(
                    log,
                    config,
                    request,
                    client,
                    url,
                    db,
                    response
                );
    }

    if(id == web::controllers::CONTROLLER_CONTACT) {
        return new web::contact::Controller(
                    log,
                    config,
                    request,
                    client,
                    url,
                    db,
                    response
                );
    }

    return nullptr;

}
```

Obrázek B.4: Ukázka továrny na kontroléry

Na závěr je důležité přidat nově vytvořené soubory do výčtu v `CMakeLists.txt`, aby byly součástí kompilace (viz ukázka B.5). Pokud je v konfiguračním souboru webové aplikace (`www/web.config`) zapnutý vývojářský mód, lze si v konzoli prohlížeče zobrazit podrobnosti o aktuálně spuštěné komponentě a další užitečné informace.

```
set(
    web
    ./src/web/routerFactory.cpp
    ./src/web/controllerFactory.cpp
    ./src/web/homepage/model.cpp
    ./src/web/homepage/view.cpp
    ./src/web/homepage/controller.cpp
)
```

Obrázek B.5: Ukázka výčtu komponent uvnitř souboru CMakeLists.txt

Příloha C

Externí přílohy

C.1 Webový framework

Webový framework je přiložen v komprimovaném souboru `annmu.zip`. Instalace a použití je popsáno v technické dokumentaci.

C.2 Ukázka použití

Ukázka použití webového frameworku je přiložena v komprimovaném souboru `ukazka.zip`. Instalace a použití je popsáno v technické dokumentaci. Zde je navíc nutné nahrát již předpřipravenou databázi `ukazka.sql` do databázového serveru.

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Novák** Jméno: **Adam** Osobní číslo: **492188**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Otevřená informatika**

Specializace: **Základy umělé inteligence a počítačových věd**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

C++ framework pro vývoj webových aplikací

Název bakalářské práce anglicky:

C ++ Framework for Web Application Development

Pokyny pro vypracování:

Cílem práce je analyzovat existující webové frameworky a navrhnout a implementovat bezpečný a rychlý webový framework v jazyce C++. Smyslem analýzy je identifikace obecné struktury webových frameworků. Výsledky práce budou průběžně testovány a na závěr ověřeny realizací webové stránky.
1. Vyberte z nabídky webových frameworků řešení, která jsou prakticky využívána a vytvořena v různých jazycích. Analyzujte vybraná řešení. Zaměřte se na vnitřní procesy, o které se webové frameworky starají a pokuste se definovat obecnou strukturu webového frameworku.
2. Definujte pojmy rychlost a bezpečnost webového frameworku. Srovnejte vybrané webové frameworky z hlediska rychlosti, zamyslete se nad bezpečností jednotlivých řešení. Odůvodněte význam využití jazyka C++ při tvorbě frameworku.
3. Navrhněte webový framework, zaměřte se na jeho bezpečnost a rychlost. Základní funkcionalitu frameworku implementujte v jazyce C++.
4. Systém otestujte vytvořením webové stránky. Popište náročnost tvorby webové stránky s využitím frameworku. Pokuste se srovnat vytvořený webový framework s analyzovanými řešeními, zaměřte se hlavně na bezpečnost a rychlost.

Seznam doporučené literatury:

[1] Nokleberg, Ch., Hawkes. B. Best Practice: Application Frameworks. Communications of the ACM, vol. 18, no. 6, pp. 42-49, 2021.
[2] Pater, J. Modern web application frameworks. Master's thesis, FI MU, Brno, 2015.
[3] Milić, M., Stanojević, V., Vlajić, S., Ognjanović, M. Guidelines for framework development process. In Proceedings of the 2011 7th Central and Eastern European Software Engineering Conference, USA, 2011.
[4] Grudl, D., Černý, H., Hůla, M., Šulc, M. Nette 3.1 Documentation. Praha, 2021. Dostupné na: https://doc.nette.org/cs/3.1/

Jméno a pracoviště vedoucí(ho) bakalářské práce:

RNDr. Ingrid Nagyová, Ph.D. kabinet výuky informatiky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **09.02.2022** Termín odevzdání bakalářské práce: **20.05.2022**

Platnost zadání bakalářské práce: **30.09.2023**

RNDr. Ingrid Nagyová, Ph.D.	prof. Ing. Tomáš Svoboda, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

