Střední průmyslová škola elektrotechnická
a Vyšší odborná škola Pardubice

**STŘEDNÍ PRŮMYSLOVÁ ŠKOLA ELEKTROTECHNICKÁ**

**MATURITNÍ PRÁCE - WEBOVÉ STRÁNKY**

# YADC - Yet Another Danbooru Clone

březen 2020 Jan Kužílek

*„Prohlašuji, že jsem maturitní práci vypracoval(a) samostatně a použil(a) jsem literárních pramenů, informací a obrázků, které cituji a uvádím v seznamu použité literatury a zdrojů informací a v seznamu použitých obrázků a neporušil jsem autorská práva.*

*Souhlasím s umístěním kompletní maturitní práce nebo její části na školní internetové stránky a s použitím jejich ukázek pro výuku.“*

V pardubicích dne ..... Podpis: .....

# Anotace

YaDc - Moderní ImageBoard pro moderní lidi. Sdílej a stahuj moderované obrázky a tapety s Anime/Manga tématikou.

Klíčová slova: imageboard, anime, manga, tapety, obrázky

# Anotation

YaDc - Modern ImageBoard for modern people. Share and download curated Anime/-Manga themed images and wallpapers.

Keywords: imageboard, anime, manga, wallpapers, pictures

# Obsah

# Úvod

Projekt umožňuje uživatelům sdílet obrázky ve formě příspěvků, které budou mít přiřazené kategoricky roztříděné tagy, věkovou přístupnost a další parametry. Před oficiálním akceptováním příspěvku bude každý obrázek patřičně zkontrolován moderátory a označen jako validní. Každý obrázek je volně přístupný ke stažení v původním formátu i zmenšené verzi formátu JPEG. Kromě toho může každý uživatel u příspěvku napsat vlastní komentář.

Cílem bylo vytvořit modulární moderovaný imageboard systém s filtrovatelným obsahem jednoho zaměření, které se dá dle instance přizpůsobit, s použitelným API, které umožní stavět aplikační klienty pro propojení s ostatními sociálními službami. Backend projektu je postavený na mikroframeworku Flask, běžícím v Pythonu, čímž je možné dosáhnout vyšší modulárnosti s možností v budoucnu implementovat další užitečné funkce.

Motivací byla převážně absence moderně vypadajícího, self-hostovatelného systému, zaměřeného na sdílení a třídění obrázků a pozadí plochy.

Frontend je moderní, přesto minimalistický, umožňující rychlé prohlížení a stahování obrázků. Je postavený s myšlenkou jednoduchosti a nezávislosti na přebytečných knihovnách jako je Bootstrap nebo jQuery.

# 1 Analýza obdobných webových stránek

## 1.1 Konachan.net

Adresa `https://konachan.net`

Konachan.net běží na projektu Moebooru[1], hluboce modifikovaném forku Danbooru[2]. Je napsán v ne-úplně standardním frameworku Ruby on Rails, který ve výsledku jenom zhoršuje rozšiřitelnost projektu. Nabízí moderovaný systém obrazových příspěvků, filtrovatelný pomocí tagů, s komentáři u každého příspěvku, rozsáhlým API a spoustou dalších funkcí. Frontend je jednoduchý, nabízející docela pohodlné prohlížení na desktopu, avšak neresponzivní, takže na mobilních zařízeních takřka nepoužitelný.

### Kladné stránky

- klasický vzhled imageboard, bez zbytečností, které by jinak zpomalovaly stránku
- standardizované API
- poměrně kompletní ekosystém (vestavěná wiki, různé „pooly“ odebírané uživateli)
- open-source

### Záporné stránky

- Ruby on Rails - špatná rozšiřitelnost
- neresponzivní vzhled, tudíž nevhodný pro prohlížení na mobilních zařízeních
- nepřehledné pro průměrného potencionálního uživatele, bez integrace se sociálními sítěmi

## 1.2 Pixiv.net

Adresa `https://www.pixiv.net`

Pixiv je kompletní, (převážně v Japonsku) populární platforma/sociální síť zaměřená na tvůrce ilustrací, mangy a knižních novel. Obsahuje žebříčky nejlepších, přizpůsobuje zobrazovaný obsah podle zájmů uživatele.

### Kladné stránky

- moderní, responzivní vzhled
- pokročilé funkce moderních sociálních sítí (např. streamování tvorby v reálném čase)

[1] `https://github.com/moebooru/moebooru`
[2] `https://github.com/r888888888/danbooru`

**Záporné stránky**

- nepřehledné rozhraní
- bez registrace nepřístupné
- proprietární

## 1.3 Deviantart.com

Adresa `https://www.deviantart.com/`

Deviantart je centralizovaná platforma pro tvůrce pro publikování své umělecké obrazové tvorby. Autor má možnost svůj obsah limitovat na určité skupiny lidí anebo ho monetizovat.

**Kladné stránky**

- moderní, responzivní vzhled
- elegantní prezentace tvorby

**Záporné stránky**

- přílišná závislost na javascriptových knihovnách, které způsobují stránku nepoužitelnou v pomalejších prostředích
- vysoká míra komercializace
- proprietární

# 2 Návrh projektu

## 2.1 Cílové skupiny

Projekt je primárně cílen na anime fanoušky a ostatní znalce moderní Japonské tvorby, kteří ocení open-source moderovanou platformu pro obrazovou tvorbu v nejvyšší/původní kvalitě. Primární využití by mělo být uchovávání/archivování tvorby do budoucna s transparentním přístupem pro kohokoli.

## 2.2 Administrace webu

Projekt má 3-úrovňovou, resp. 4-úrovňovou administraci. Jsou zde normální uživatelé, kteří jen pasivně konzumují obsah a nemají proto přístup nikam, kromě nastavení vlastního profilu. Pak zde jsou tvůrci – uploadeři, kteří už nějak přispěli do databáze a mají tak veřejný status přispěvatele, ale navíc se jim uvolní pouze správa vlastních příspěvků v management prostředí. Dále zde jsou už zvolení moderátoři, kteří mají práva měnit příspěvky, přidávat, upravovat nebo mazat tagy a moderovat komentáře. Nemají však možnost komentáře definitivně mazat, takže uživatel má vždy možnost obsah upravit, aby komentář zase moderátor odblokoval. Admin má pak absolutní práva. Jediné jeho restrikce jsou neoprávněné změny uživatelských nastavení včetně jeho uživatelského jména.

## 2.3 Databáze

Databáze se skládá ze 4 hlavních tabulek. Je tu tabulka `user` v níž jsou uchovávány veškeré uživatelské informace, jako uživatelské jméno, email, hash hesla, jeho status oprávnění, status účtu a samozřejmě informace o jeho profilu s preferencí nejvyšší kategorie přístupnosti. Za pozornost stojí relace s tabulkou `tags`, identifikující blacklist všech tagů, které uživateli nevyhovují a nechce je ve výpisu zobrazovat.

Jako druhý nejhlavnější prvek tvoří tabulka `post`, která má také navázáno nemalé množství relací. Obsahuje prakticky veškeré informace o nahraném obrázku jako např. md5 hash pro zajištění unikátnosti souboru, datový typ souboru, rozměry a velikost souboru jako užitečná informace pro uživatele, aby nebylo zapotřebí tyto parametry generovat během zpracování dotazu. Jako vyžadovaná informace, sloužící pro ověření pravosti a věrohodnosti během moderace příspěvku, je zde pak zdroj – odkaz na stránku autora, nebo jinou stránku z níž mohl uživatel čerpat.

Jsou zde definované 3 základní kategorie přístupnosti. „Safe“ jako bezpečný pro všechny věkové kategorie, „Questionable“, pro obrázky které nevypadají úplně akceptovatelně, ale zase nepatří na 100% do kategorie 18+ a pak „Explicit“, který není v žádném případě vhodný pro nezletilé osoby. Kategorie se rozlišuje subjektivně podle daného kontextu, ale vždy by se mělo dosáhnout podobného výsledku.

Každý příspěvek má svého uploadera v podobě instance typu `user` a popř. approvera - moderátora, který zodpovídá za stav příspěvku. Další relace odkazuje na „parent“ `post`, který by měl být nějakým způsobem/kompozicí příbuzný tomuto. Tímto se dostáváme

k `m:n` relaci (využívající vázací tabulku `post_tags`) s tabulkou `tags`, která definuje každý tag, který by měl daný obrázek vystihovat.

Tabulka `tags` je jednoduchá. Je definována unikátním názvem tagu a jeho kategorií. Tagových kategorií je přesně 6. Vyjadřují typ tagu ve vztahu s obsahem příspěvku tak, aby to v budoucnu umožnilo jednodušší parsování databáze, ale i vizuálního rozlišení.

Kromě tagů je k tabulce `post` i `user` připojená tabulka `comment`, implementující uživatelské komentáře u příspěvku. Obsahuje vlastní obsah zprávy a vlastnost deleted, pokud byl komentář zablokován moderátorem.

## 2.4 Design a responzivita

Jedna z původních myšlenek bylo zachovat klasický imageboard formát, ale přinést trochu vylepšení v podobě responzivity a moderních designových prvků. Veškeré styly jsou zde řešené pomocí CSS, resp. SCSS stylovacího jazyka a miniaturní knihovny `include-media`[3], starající se o elegantnější řešení rezponzivity za pomoci CSS Media Queries.

```
$breakpoints: (tablet: 560px, desktop: 900px);
@include media(">=desktop") {
    ...
}
@include media("<desktop") {
    ...
}
```

Listing 1: Příklad využití knihovny include-media

Rezponzivita je rozdělena do tří rozměrů, kde minimální šířka pro desktop je 900px a pro tablet 560px. Při přechodu z desktopového rozvržení na tabletové se obsah pravého postranního panelu přesune nad hlavní obsah a při dalším zmenšování na mobilní rozvržení se kompletně skryjí odkazy v horním navigačním panelu a jsou dostupné přes menu tlačítko vpravo nahoře. Naopak při nadměrném zvětšování je tu limit pro šířku hlavního wrapperu 1300px, takže bude obsah vždy vycentrován na očích uživatele.

Stránky jsou designované v tmavém šedivém barevném schématu, ne jen kvůli tomu, že se s tmavými prvky lépe pracuje, ale i proto, že méně svítící obrazovka také méně unavuje oči uživatele. Původní design byl postaven okolo tmavě červené, ale od toho se upustilo kvůli nevhodnosti kombinace s dalšími barevnými prvky. Barvy, včetně pár dalších parametrů, lze však kdykoli jednoduše přizpůsobit přenastavením globálních proměnných v scss souboru `assets/css/main.scss`.

Hlavní indexová stránka obsahuje dynamicky se přizpůsobující výpis náhledů posledních nahraných obrázků, využívající speciálně postavený blok s `display: flex;` a `flex-flow: ro`
Každý náhledový obrázek pak má speciální za běhu generovaný atribut `flex: * 1 *px;`
(kde `*` identifikuje pixelovou šířku v poměru s minimální definovanou výškou), který zajišťuje, že se normalizuje jeho pixelová výška a budou se zobrazovat a zalamovat bez zbytečných mezer, tak jak mají. Přes veškerou snahu strávenou na tvorbě tohoto responzivního řešení, ve snaze napodobit web DeviantArt na str. 10, ale s využitím čistého

[3] `https://eduardoboucas.github.io/include-media/`

```
$bg-color: #222;
$text-color: #fff;
$a-color: #bbb;
$ahover-color: #909090;
$notif-bg-color: #000c;
$notif-bg-error: #500c;

$main-wrap-padding: 8px;
$sidepanel-width: 18rem;//14rem;
$max-content-width: 1300px;
```

Listing 2: Definice proměnných v `main.scss`

CSS v kontrastu s jejich JavaScript řešením, tento způsob není dokonalý. Například obrázkům, postaveným na výšku, nedává vždy tolik zobrazovacího prostoru, takže jsou efektivně mnohem menší. Z toho je také odvozené, že pokud uživatel na vstupu nahraje obrázek s příliš exotickým poměrem stran, je možné, že se layout výpisu bude chovat nepředvídatelně. Aby se náhledy v posledním řádku na stránce neroztáhly na celou šířku, je tu pro jistotu přidaný pseudoelement `::after` zajišťující smrštění náhledů na levý bok. V tabletovém a dále mobilním rozvržení se pomocí záporného marginu

```
section.post-list {
    @include media("<tablet") {
        margin-left: -#{$main-wrap-padding};
        margin-right: -#{$main-wrap-padding};
    }
    .posts {
        $figure-margin: 8px;
        @include media("<desktop") {
            margin-left: -#{$figure-margin};
            margin-right: -#{$figure-margin};
        }
        &::after {
            content: "";
            flex: 10000 0 350px;
        }
        > figure {
            margin: $figure-margin;
            ...
        }
    }
}
```

Listing 3: Zjednodušený pohled do designu výpisu náhledů

odeberou okraje u náhledů pro lepší uživatelský experience. Výpis dále využívá `srcset` HTML atribut pro poskytnutí náhledu v rozlišení adekvátním pro dané zařízení.

V levém panelu se nachází speciální prvek pro přehled a výběr/filtrování pomocí tagů. Výpis, který po najetí myší umožní následný výběr pro filtraci, indikuje tagy, které se v tento moment vyskytují v příspěvcích na přehledové části vpravo a jejich současný unikátní počet. V horní části je vyhledávací pole, kterým lze vyhledávat a vybírat tagy, které nejsou na stránce.

Tento prvek s tagy je přizpůsoben, aby byl znovuužitelný i na jiných místech webu (např. na stránce uploadu nebo úpravy příspěvku) jako náhrada za multiselect. Prvek

je použitelný i v prostředích s vypnutým JavaScriptem, kde se zobrazí jen jako textový input obsahující mezerou oddělené hodnoty. Tagy nabízené na hlavní stránce se v tomto prostředí chovají jako jednoduché odkazy, takže se výpis při kliknutí automaticky aktualizuje. Očividně však není možné provádět hledání s nápovídáním.

Stránka zobrazení příspěvku je neméně zajímavá - kromě postranního panelu obsahuje flexový wrapper pro JPEG sample obrázku ve vyšším rozlišení a komentářovou sekci. Při přechodu na tabletové rozvržení je díky funkci `display: contents;`, který předá své položky bloku o třídu výše, možné vložit boční panel mezi náhled obrázku a komentářovou sekci.

```
.important-subwrap {
    display: flex;
    flex-flow: column nowrap;
    flex-grow: 1;

    @include media("<desktop") {
        display: contents;
    }
}
```

Listing 4: Řešení responzivity na stránce příspěvku

Původní design tohoto přeskládání byl postaven poněkud složitějším způsobem. Vlastní náhled a sidebar byl ve svém flexboxu a postranní panel měl `height: 0;`, kde jeho obsah přetíkal ven. Když se nastavil patřičný margin na levé straně, tak to na desktopu umožňovalo umístit komentáře rovnou pod náhled a v mobilním rozvržení naopak dát náhledu `order: -1;` a připíchnout ho tak na vrch stránky. Nevýhoda tohoto řešení spočívala v překrývání levého panelu s patičkou stránky, takže se od tohoto designu upustilo.

Levý panel zde obsahuje kromě výpisu tagů, které definují daný příspěvek, s jejich globálním počtem výskytů i samotný popis příspěvku a malý panel pro editaci příspěvku. Popis obsahuje výpis prakticky veškerých parametrů příspěvku, které se vyskytují v tabulce `post` (str. 11), včetně lidsky formátovaného relativního času nahrání a jmen uploadera a moderátora, který přispěvek schválil (popř. statusu schválenosti). Může se zde vyskytovat odkaz na „parent“ příspěvek, pokud současný nějaký má, nebo naopak jeho děti. Pod položkami jsou odkazy na originální zdrojový soubor beze změny, popřípadě i JPEG verzi, pokud byl originál ve formátu PNG.

Komentářová sekce má jednoduchý vzhled. Hlavička komentáře obsahuje uživatelské jméno autora a na pravou stranu odsazený control tooltip, kterým lze spravovat daný komentář. Pokud jste autor daného komentáře, po kliknutí na `edit` odkaz se obsah komentáře nahradí za textové pole pro úpravu a v tooltipu přibydou možnosti pro aktualizaci, nebo zrušení změn. Jako moderátor se zde objeví odkazy pro zablokování/odblokování komentáře. Pokud bude mít uživatel vypnutý JavaScript, panel úprav se zobrazí automaticky při najetí kurzorem nad blok komentáře. Pod hlavičkou je samotný obsah komentáře. V případě zablokování komentáře moderátorem, se jeho obsah nahradí červenou poznámkou o jeho stavu zablokování. To však neznemožní jeho správu, jelikož pro autora i moderátora bude nadále obsah zprávy viditelný.

Každý uživatel má svou stránku profilu. Jsou zde veřejně dostupné statistiky o jeho

užívání služby, blacklist tagů a seznam posledních komentářů. Jako přihlášený uživatel lze najít svůj profil kliknutím na položku `Profile` v padacím menu, dostupném po najetí kurzorem na jméno uživatele v horním panelu. Jako registrovaný uživatel máte také přístup do stránek administrace. Na to je tu položka `Settings`.

Jako normální uživatel můžete manipulovat s nastavením svého profilu. Jsou zde formuláře pro úpravu vlastní biografie, změnu hesla nebo emailové adresy, nastavení preferované maximální přístupové kategorie a blacklistu tagů příspěvků zobrazovaných na hlavní stránce a samozřejmě možnost smazat veškerá svá uživatelská data.

Moderátorům v levém navigačním panelu přibydou položky s odkazy na stránky pro hromadnou správu detailů příspěvků a tagů. Pokud uživatel už přispěl nějakým obrázkem do databáze, může se přes levý panel dostat do správy příspěvků také. Zobrazí se však pouze příspěvky které sám přidal. Rozhraní pro správu je ve formě základní tabulky parametrů. Na konci každého řádku ve sloupci `Manage` kliknutím na ikonku pro editování nebo jednoduše poklikáním na položku, kterou chcete upravit, lze zobrazit formulář pro daný řádek s ikonkami pro zahození a odeslání změn. Vždy je na konci také ikonka koše pro vymazání daného záznamu. Stránka správy tagů navíc umožňuje vytváření nových tagů. Responzivita u tabulky není bez použití přebytečných knihoven elegantní na implementaci, proto jsem se zdržel přílišných složitostí a aplikoval horizontální scrollování, aby alespoň tak bylo možné na mobilních platformách provádět modifikace.

Administrátorům dále přibyde možnost pro správu uživatelů.

# 3 Popis projektu

Ke stavbě svého projektu jsem využil mikroframework Flask, který běží na jazyku Python. S mými dřívějšími zkušenostmi s jazykem Python a frameworkem Flask, jsem se rozhodl, že využiji příležitosti a prohloubím své znalosti na větším projektu. Flask není plnohodnotný framework typu Django nebo Laravel v případě PHP, ale umožňuje jednoduché rozšíření svých funkcí pomocí spousty pomocných balíčků vytvořených komunitou.

Jako databázový engine používám PostgreSQL. Moje volba byla postavena na vyšší jednoduchosti obsluhy a nižší velikosti/výkonové náročnosti, a tím samozřejmě odvozené vyšší bezpečnosti celého engine ve srovnání např. s MySQL/MariaDB SQL DBMS implementací.

## 3.1 Frontend

Flask má vestavěný šablonový engine Jinja2, pomocí něhož jsem schopen efektivně třídit a skládat pohledy s přidanou vrstvou bezpečnosti, jako ochrana před XSS escapováním znaků. Tímto jsem schopen vytvořit základní kostru stránky, ze které lze dále dědit formou bloků.

```
{% extends 'layout/base.html' %}
{% block content %}
<section class="sidepanel">
    {% block sidebar %}{% endblock %}
</section>
<div class="important-subwrap">
    {% block main_content %}{% endblock %}
</div>
{% endblock content %}
```

Listing 5: Příklad jednoduché šablony, použité pro postranní panel

Základní šablona se nachází v souboru `templates/layout/base.html`. Obsahuje základní HTML strukturu včetně hlavičky, v níž se importují zabundlované (viz. Backend) CSS soubory a `Font-Awesome` icon pack, který využívám na mnoha místech projektu. Tělo stránky obsahuje základní strukturu hlavního navigačního panelu, cyklus pro výpis pop-up notifikací a patičku stránky s importy opět zabundlovaných JS skriptů. Jelikož z této šablony dědí všechny stránky projektu, budou tyto prvky dostupné všude.

V mobilním rozvržení se v hlavičce aktivuje skript `assets/js/base.js`, který se postará o zobrazení přetékajícího menu a o zmrazení scrollování stránky.

Lze si také všimnout, že zde, stejně jako na mnoha stránkách frontendu, využívám pomocné funkce z modulu `utils.py` (str. 19), které jsou do tzv. (frontendového) kontextu naimportované v `__init__.py`.

Většina stránek dědí i ze `layout/base_sidebar.html`, který přidává postranní panel, ale zdaleka zajímavější je `layout/base_sidebar_tags.html`. Je zde vidět základní struktura panelu s tagy. Většina interaktivních prvků je však definována až ve skriptu v

`assets/js/taginput.js`. Tento skript se aplikuje na každý prvek se vstupem pro tagy. Po zadání příslušného slova do textového inputu lze kliknutím na nabízený tag v popupu nebo stisknutím klávesy `<Enter>` zvýrazněný tag přidat do vybraných. Na hlavní stránce se opakovaným stisknutím `<Enter>` potvrdí vyhledávání a po pravé straně se aktualizuje výpis náhledů příspěvků. V popupu nabízených tagů během vyhledávání se lze také pohybovat pomocí klávesových šipek nebo kombinací `<Tab>` a `<Shift-Tab>`. Vyhledávání probíhá pomocí AJAX dotazů v pozadí využívající `Fetch` API. Dotaz se provede vždy v určitém časovém intervalu poté, co uživatel přestane psát.

Template `layout/management.html` obsahuje univerzální znovuužitelnou strukturu pro tvorbu tabulek pro management, který masivně využívá schopností Jinja2 engine. Je tu předdefinovaná kostra tabulky s formuláři pro každý řádek, které se pomocí `for` cyklu generují. Dědící šablony z adresáře `manage/` pak mají díky block definicím možnost vložit vlastní obsah hlavičky a polí do těla tabulky. Pole jsou generována pomocí makra, `genfield()`, kterému lze pomocí `call` statementu vložit vlastní payload pro lepší přizpůsobení zobrazovaného pole, když není editováno (např. jako odkaz na příslušnou stránku příspěvku).

```
{% macro genfield(formfield=None) %}
{% if not formfield %}
<td>
    <span>{{ caller() }}</span>
</td>
{% else %}
<td>
    <span class="notedit">{{ caller() }}</span>
    <span class="edit">{{ formfield() }}</span>
</td>
{% endif %}
{% endmacro %}
```

Listing 6: Makro `genfield()` v `management.html`

Hlavní template pro indexovou stránku v `post/index.html` obsahuje pouze jednoduchý výpis náhledů příspěvků. Adresa na zdroj obrázku je generována metodou `Post.url()`, která vždy vrátí správný formát podle `IMAGE_STORE` enumerátoru uvedeného jako argument.

Výpis je v backendu limitován, proto je zde také import pro makro `render_pagination()` z `_includes.html`, které využije poskytnutého objektu `Pagination` z dotazu na straně backendu. Implementovaný stránkovací mechanizmus umožňuje kompletní volnost v pohybu vpřed a vzad, poskytující i odkazy na sousední strany.

Stránka pro zobrazení jednoho příspěvku kromě postranního výpisu jeho parametrů a jednoduchého formuláře pro úpravu, obsahuje i komentářový výpis. Opět se zde v cyklu pro každý poskytnutý komentář volá makro ze soboru `_includes.html`, které tentokrát vrátí kompletní blok komentáře.

Stránka s formulářem pro upload příspěvků je zajímavá integrací dříve využitého tagového inputu, opět definovaného uvnitř `_includes.html`. Toto makro `render_tag_input()` jako argument vezme jednoduché `wtforms` textové pole a přetvoří ho na přizpůsobený prvek, ovládaný skriptem `taginput.js`.

## 3.2 Backend

Základem celého projektu je tzv. app-factory v souboru `__init__.py` v kořenové složce projektu, kde se definuje chod celé Flask aplikace.

Nejprve se zde načtou globální konfigurační proměnné ze souboru `config.py` v instančním adresáři repozitáře. Konfigurace je pak dostupná pod objektem `config` v instanci Flasku.

Dále se zde inicializují balíčky jako např. SQLAlchemy[4] ORM (nebo spíše jeho adaptace pro Flask – `Flask-SQLAlchemy`[5]), který využívám pro operace s databází nebo `Flask-Assets`[6], který se postará o kompilaci a minimalizaci SCSS a zabalení do jednotných souborů, a to i v případě JS.

### 3.2.1 Struktura aplikace

#### Směrování

Flask obsahuje vlastní směrovací systém s podporou tzv. blueprintů, jimiž jsem schopen přehledně oddělit specifické části projektu do zvláštních souborů a přidělit jim vlastní prefix v URL cestě. Uvnitř blueprintů je pak možné pomocí tzv. funkčních dekorátorů označit koncové funkce cestami nebo jinými podmínkami.

Program je rozdělen do pěti blueprintů.

Z uživatelské perspektivy je nejpodstatnější částí blueprint `bp/post.py`, kde jsou definované endpointy pro zobrazení indexu a jednotlivých příspěvků, ale i formulář pro nahrávání příspěvků, koncový bod pro uživatelské komentáře, jednoduché API pro automatické napovídání během výběru tagů k filtraci a velice jednoduché API pro aplikaci třetí strany, aby mohla přistupovat k obrázkové databázi.

```
from yadc.bp import main, post, auth, manage, user
app.register_blueprint(main.bp)
app.register_blueprint(post.bp, url_prefix='/post')
app.register_blueprint(auth.bp, url_prefix='/auth')
app.register_blueprint(manage.bp, url_prefix='/manage')
app.register_blueprint(user.bp, url_prefix='/user')
```

Listing 7: Inicializace blueprintů v `__init__.py`

Neméně důležitý je pak `bp/main.py`, který má na starost zpracovávat dotazy na obrazová data uložená ve složce instance. Jsou zde koncové body pro pět různých formátů/velikostí každého nahraného obrázku s dynamicky parsovanou URL cestou.

V souboru `bp/manage.py` jsou endpointy pro stránky managementu a jejich formuláře a `bp/user.py` obsahuje koncové body pro uživatelský profil s jeho stránkou nastavení.

Blueprint `bp/auth.py` se nakonec stará o přihlášení a registrace.

[4] `https://www.sqlalchemy.org/`
[5] `https://flask-sqlalchemy.palletsprojects.com/en/2.x/`
[6] `https://flask-assets.readthedocs.io/en/latest/`

### Formuláře

Pro zelegantnění práce s formuláři napříč projektem využívám Python balíček `wtforms`. Ten umožňuje snadnou definici univerzálních formulářů s možností validace polí a případné reportování nalezených chyb. Má také částečně integrovanou ochranu před CSRF útoky pomocí skrytého formulářového pole.

```
class LoginForm(CSRFForm):
    username = StringField('Username', validators=[DataRequired()],
    ↪  render_kw=dict(placeholder="Username"))
    password = PasswordField('Password', validators=[DataRequired()],
    ↪  render_kw=dict(placeholder="Password"))
    remember_me = BooleanField('Remember me')
    submit = SubmitField('Log In')
```

Listing 8: Implementace přihlašovacího formuláře s použitými validátory

Definici všech užitých formulářů lze nalézt v souboru `forms.py`

### Přístup do databáze

Do databáze přistupuji pomocí SQLAlchemy Object Relation Mapperu. Mohu si tak definovat tabulky přímo jako třídy a manipulovat s nimi pomocí třídy SQLAlchemy. Pro jednoduchý SELECT všech záznamů modelu `Post` mi postačí zavolat `Post.query.all()` (zkratka pro `db.session.query(Post).all()`) a vrátí se mi list instancí objektu `Post`, s nímiž mohu dále manipulovat.

Definice pomocí modelů mi umožňuje definovat vlastní metody na daném objektu pro provádění užitečných operací. Výborným příkladem je funkce generující `flex` CSS atribut pro náhledy ve výpisu na hlavní stránce (na str. 12).

V souboru `models.py` se kromě modelů vyskytují i enumerace obecných hodnot jako je datový formát nahraných obrázků, úrovně oprávnění uživatelů nebo kategorie tagů. Se všemi dokáže SQLAlchemy operovat.

### Utility

Pro uchování užitečných funkcí, které však nepatří na žádné specifické místo a je potřeba k nim přistupovat v globálním měřítku, slouží soubor `utils.py`. Jsou zde funkce napomáhající s manipulací s URL adresami jako např. nahrazování parametrů, tvorba adres pro tagy nebo jednoduché *„flashování“* errorů z formulářových endpointů, aby se poté mohly vypsat uživateli. Je zde také definovaná vypůjčená funkce `sizeof_fmt()` [7], která převádí jednotky velikosti souboru.

### Konfigurace

Příklad základní konfigurace je uložen ve skriptu `config.def.py`. Před nasazením projektu je však nutné tento soubor přesunout do instančního adresáře v kořenu repozitáře pod jménem `config.py`, jinak nebude aplikace operovat správně. V konfiguračním souboru lze najít základní parametry, jako je URI pro přístup do databáze obsahující

[7] `https://web.archive.org/web/20111010015624/http://blogmag.net/blog/read/38/Print_human_readable_file_size`

přihlašovací údaje, adresu a název databáze nebo náhodný tajný klíč, určený pro generování bezpečných tokenů a hesel napříč aplikací. Lze zde také najít vlastní konfigurační proměnné měnící např. počet zobrazovaných náhledů na hlavní stránce, položek na stránkách managementu nebo možnost pro změnu názvu instance, který se projeví na mnoha místech aplikace.

### 3.2.2 Hlavní stránka

Ačkoli se to nemusí zdát, struktura endpointu pro vypisování příspěvků není jednoduchá. Zpracovávají se zde vstupní parametry URL v podobě tagů a vynucené hodnoty věkové přístupnosti. Pokud byl věkový rating vynucen v URL, instantně se převede na enumerátor `RATING` a vloží se pro příští návštěvy do `session`, aby byl při příštích návštěvách indexu dostupný. Pokud není vynucen, tak se jako default nastaví preference přihlášeného uživatele. Pokud nejsou splněny podmínky výše, zobrazí se jen obsah s bezpečnou tématikou. Věková přístupnost se pomocí funkce `matched` na instanci enumerátoru počítá sestupně, takže není možné vypínat jednotlivé kategorie.

Poté co se provede implicitní SELECT tagů, které má uživatel v blacklistu, se dynamicky vytvoří dotaz a pomocí speciální `paginate()` funkce se dotaz provede a převede na objekt `Pagination`, v němž se dá později stránkovat a současně je efektivnější než `OFFSET` a `LIMIT` atributy.

```
posts_query = Post.query
if f_tags or blacklist_tags:
    posts_query = posts_query.join(Post.tags).group_by(Post.id)
    if blacklist_tags:
        subquery = db.session.query(Post.id)
            .join(Tag.posts).filter(Tag.content.in_(blacklist_tags)).subquery()
        posts_query = posts_query.filter(~Post.id.in_(subquery))
    if f_tags:
        posts_query = posts_query.filter(Tag.content.in_(f_tags))
            .having(func.count(Post.id)==len(f_tags))
posts_query = posts_query.filter(Post.rating.in_(m_ratings))
    .order_by(Post.created.desc())
posts = posts_query.paginate(page, current_app.config.get('POSTS_PER_PAGE'))
```

Listing 9: Dynamická tvorba databázového dotazu v závislosti na vstupu

### 3.2.3 Přihlášení a registrace

Mezi jeden z nejvíce užitečných rozšíření pro Flask je `Flask-Login`[8]. Zajistí integraci s modelem uživatele a postará se i o session proměnné. Na mě byla jen integrace s modelem v podobě tvorby a srovnávání hesel funkcemi `Post.create_password()` a `Post.check_password()` v podobě `PBKDF2` hashů.

Jak již bylo zmíněno, endpointy pro přihlašování a registrace jsou v blueprintu `bp/auth.py`. Ve většině případů jde o velice podobné zpracování dotazů, kde se při `POST` requestu zvaliduje odeslaný formulář a provede odpovídající akce, nebo jen vrátí zpracovaný template, v němž se nová instance formuláře vyrenderuje.

[8] `https://flask-login.readthedocs.io/en/latest/`

```
def create_password(self, password):
    self.pass_hash = generate_password_hash(password, salt_length=16)

def check_password(self, password):
    if self.pass_hash is None:
        return False
    return check_password_hash(self.pass_hash, password)

def login(self, remember):
    login_user(self, remember=remember)
    self.last_login = utcnow()
```

Listing 10: Metody na třídě `User` pro práci s hesly a operaci s `Flask-Login`

```
@bp.route('/login', methods=['GET', 'POST'])
def login():
    if fl.current_user.is_authenticated:
        return redirect(url_for('main.index'))

    form = LoginForm(request.form)
    if request.method == 'POST' and form.validate():
        user = User.query.filter_by(username=form.username.data).first()
        if user is None or not user.check_password(form.password.data):
            flash('Invalid username or password.', category='error')
            return redirect(url_for('.login'))
        user.login(remember=form.remember_me.data)
        db.session.commit()

        flash(f'Logged in as {user.username}.')
        return redirect(nextpage())

    flasherrors(form)
    return render_template('auth/login.html', form=form)
```

Listing 11: Endpoint pro login uživatele

Je zde implementována registrace, přihlášení, odhlášení a reset hesla.

### 3.2.4 Nahrávání příspěvků

Upload příspěvku je, jak již je zřejmé, dělen do několika částí. Nejprve se za pomoci balíčku `Flask-Login` a jeho dekorátoru `@loginrequired` zajistí, že je uživatel přihlášen, jinak bude přesměrován na stránku s přihlášením. V případě, že uživatel pomocí POST dotazu odesílá obsah formuláře, proběhne validace na straně formuláře, a při jakémkoli erroru ho přesměruje na původní stránku a notifikací ho upozorní na chybnost odeslaných dat.

Kromě základní validace textových vstupů se zde kontroluje i stav nahraného obrázku. Nejprve se za pomoci balíčku `Magic` srovná reálný mimetype souboru obrázku s podporovanými formáty, posléze se pomocí knihovny `Pillow` zkontroluje integrita souboru. Poskytnutím souboru statické metodě `Post.fileinfo()` se dále obstarají metadata o daném obrázku, na kterých se dále kontroluje unikátnost a minimální rozlišení nahrávaného příspěvku. Po úspěšné validaci se objekt `Post` inicializuje, originál obrázku se uloží do složky v adresáři instance a zavoláním `Post.generate_image_files()` se

```
fileinfo = Post.fileinfo(file)
post = Post.query.filter_by(md5=fileinfo['md5']).first()
if post is not None:
    raise ValidationError('Image with same hash already exists. Reposting is not
    ↪  allowed.')

width, height = fileinfo['resolution']
if width*height<min_width*min_height or width<min_size or height<min_size:
    raise ValidationError('Image has too low resolution.')
```

Listing 12: Část validace nahrávaného souboru

opět pomocí balíčku `Pillow` vygenerují všechny jeho varianty pro pozdější užití.

### 3.2.5 Management

Blueprint pro management v `bp/manage.py` má poněkud jinou strukturu než autorizační koncové body nebo endpoint pro upload. Pro zpřehlednění a zjednodušení manipulace jsou zde oddělené cesty pro zobrazení spravovacích tabulek dotazy GET a ty pro formuláře dotazem POST. I ve `forms.py` si lze povšimnout abstraktní třídy `EditForm`, která definuje základní způsob manipulace s daty a z níž jednotlivé typy dědí.

```
class EditForm(CSRFForm):
    id = HiddenField('ID')

    create = SubmitField('Create')
    edit = SubmitField('Modify')
    delete = SubmitField('Delete')

    def validate_id(form, field):
        # if (form.edit.data or form.delete.data) and not field.data:
        if not form.create.data and not field.data:
            raise ValidationError('ID must be defined to be able to modify.')
```

Listing 13: Abstraktní třída formuláře `EditForm`

V endpointech pro zpracování poskytnutých dat tak lze podle jména odeslaného pole typu `submit` rozhodovat, která operace se provede. Kromě užitých dekorátorů `@loginrequired` jsou i zde použity i dekorátory `@moderatorrequired` a `@adminrequired`, implementované v modulu s modely. Je zde kromě bodů pro modifikaci příspěvků, tagů a uživatelů umístěný i endpoint pro moderaci komentářů, která je však zatím dostupná pouze ze stránky příslušného příspěvku.

# 4 Závěr