

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA STAVEBNÍ
PROGRAM GEODÉZIE A KARTOGRAFIE
OBOR GEOMATIKA



DIPLOMOVÁ PRÁCE
TVORBA WEBOVÉ APLIKACE PRO PROJEKT VISKALIA

Vedoucí práce: Ing. Martin Landa, Ph.D.
Katedra geomatiky

červen 2021

Bc. Adam KULHAVÝ

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: <u>Kulhavý</u>	Jméno: <u>Adam</u>	Osobní číslo: <u>440829</u>
Zadávající katedra: <u>Katedra geomatiky</u>		
Studijní program: <u>Geodézie a kartografie</u>		
Studijní obor: <u>Geomatika</u>		

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce: <u>Tvorba webové editorské aplikace pro projekt Viskalia</u>	
Název diplomové práce anglicky: <u>Web application for Viskalia project</u>	
Pokyny pro vypracování: Cílem diplomové práce je vytvořit webovou aplikaci za použití frameworku Django. Hlavním účelem této aplikace je správa a editace zájmových objektů výběrové databáze projektu Viskalia. Což zahrnuje též vizualizaci přidruženého multimediálního obsahu. Administrátorská část aplikace bude umožňovat správu uživatelů, jejich rolí a oprávnění. Uživatelská práva budou vztažena na úroveň jednotlivých zájmových objektů.	
Seznam doporučené literatury: Mastering Django - Nigel George, ISBN 9781787286344 Django 3 By Example: Build powerful and reliable Python web applications from scratch - Antonio Mele, ISBN 1838981950 Python 3: Výukový kurz - Mark Summerfield, ISBN 9788025127377	
Jméno vedoucího diplomové práce: <u>Ing. Martin Landa, PhD.</u>	
Datum zadání diplomové práce: <u>19.2.2021</u>	Termín odevzdání diplomové práce: <u>17.5.2021</u> <i>Údaj uveďte v souladu s datem v časovém plánu příslušného ak. roku</i>
Podpis vedoucího práce	Podpis vedoucího katedry

III. PŘEVZETÍ ZADÁNÍ

Beru na vědomí, že jsem povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je nutné uvést v diplomové práci a při citování postupovat v souladu s metodickou příručkou ČVUT „Jak psát vysokoškolské závěrečné práce“ a metodickým pokynem ČVUT „O dodržování etických principů při přípravě vysokoškolských závěrečných prací“.

Datum převzetí zadání	Podpis studenta(ky)
-----------------------	---------------------

ABSTRAKT

Tato diplomová práce se zabývá vytvořením webové aplikace pro projekt Viskalia. Webová aplikace je tvořena v Python frameworku Django s připojením k databázovému systému MariaDB. Aplikace je tvořena v administrátorském prostředí, kde je optimalizována dle potřeb projektu. Jedním z cílů je také automatizované nasazení aplikace a databáze v izolovaném prostředí.

KLÍČOVÁ SLOVA

Python, Django, Webová aplikace, Docker, MariaDB, Viskalia

ABSTRACT

This Master's thesis deals with the creation of a web application for the Viskalia project. The web application is created in Python framework named Django with a connection to the MariaDB database. The application is developed in an administration environment adapted to the needs of the project. One of the goals is to create an automated deployment for application and database in an isolated environment

KEYWORDS

Python, Django, Web application, Docker, MariaDB, Viskalia

PROHLÁŠENÍ

Prohlašuji, že diplomovou práci na téma „Tvorba webové aplikace pro projekt Viskalia“ jsem vypracoval samostatně. Použitou literaturu a podkladové materiály uvádím v seznamu zdrojů.

V Praze dne

.....

(podpis autora)

PODĚKOVÁNÍ

Tímto bych rád poděkoval panu Ing. Martinu Landovi, Ph.D. za vedení diplomové práce a odbornou pomoc při jejím zpracování. Dále bych chtěl poděkovat celé své rodině za podporu a trpělivost.

Obsah

Úvod	9
1 Rešerše	10
1.1 Volba metody pro tvorbu webové aplikace	10
1.2 Volba Python frameworku	10
1.3 Volba databáze	11
1.4 Automatizace nasazení aplikace	12
1.5 Normalizace databáze	13
1.6 Objektově relační mapování	13
2 Použité technologie	14
2.1 Django	14
2.1.1 Django - instalace a inicializace projektu	15
2.1.2 Django - tvorba aplikace	17
2.1.3 Django - práce s databází	18
2.1.4 Django - přídatné balíčky	19
2.1.5 Django - administrátorské rozhraní	19
2.2 Docker	20
2.3 Git	20
2.4 MariaDB	21
2.5 phpMyAdmin	21
3 Seznámení s aplikací a databází	22
3.1 Navázání na projekt FGIS	22
3.2 Zkušební databáze	22
3.3 Centrální databáze	23
4 Tvorba aplikace	24
4.1 Vytvoření základní aplikace	24
4.2 Tvorba aplikace v uživatelském prostředí	24
4.3 Přidání obrázků	27
4.4 Statické soubory	28
4.5 Nastavení jazyka a času	28

4.6	Administrátorské prostředí	29
4.7	Oprávnění k objektům	30
4.8	Nastavení zobrazení dat a jejich vyhledávání	31
4.9	Použití balíčku admin_interface	32
4.10	Tvorba databáze	32
4.11	Doplnění výběrové databáze	33
4.12	Automatizace nasazení aplikace a databáze	34
4.13	Úprava admin prostředí	36
	Závěr	37
	Seznam zkratk	38
	Odkazy	39
	Seznam příloh	41
	A Průvodce aplikací	42
	B Zdrojový kód	46

Seznam obrázků

1.1	Flask vs Django	10
2.1	Django Logo	14
2.2	Model příklad	18
3.1	Náhled aplikace vytvořené v projektu FGIS	22
3.2	Náhled dat z centrální databáze s viditelnými duplicitami	23
4.1	Náhled views.py a jeho tříd	25
4.2	Náhled urls.py v adresáři aplikaci	26
4.3	Náhled HTML souboru edit detail	26
4.4	Náhled tabulky CvutImages pro ukládání obrázků	27
4.5	Náhled projektového urls.py	27
4.6	Náhled projektového adresáře	28
4.7	Registrace modelů v admin.py	29
4.8	Vytvoření managers v modelu aplikace	30
4.9	Úprava práv k objektům	30
4.10	Zobrazení dat a vyhledávání	31
4.11	Vyhledávání podle filtrů	31
4.12	Struktura normalizované databáze	32
4.13	Struktura výběrové databáze	33
4.14	Postup tvorby aplikace a databáze	35
4.15	Zobrazení fotografií v administrátorském prostředí	36
4.16	Vytvoření náhledu v admin.py	36
A.1	Přihlašování do administrátorské aplikace	42
A.2	Hlavní stránka pro přihlášení jako administrátor	42
A.3	Zobrazení stránky pro uživatele	43
A.4	Hlavní stránka pro přihlášení jako administrátor	43
A.5	Zobrazení záznamu s možností měnit práva uživatelů	43
A.6	Zobrazení záznamu s možností měnit práva uživatelů	44
A.7	Správa uživatelů	44
A.8	Tvorba skupin	45
A.9	Nastavení vzhledu stránky	45

Úvod

Cílem práce bylo vytvořit webovou aplikaci pro projekt Viskalia (Virtuální skansen lidové architektury), identifikační kód projektu: DG20P02OVV003. Projekt se zaměřuje na záchranu fondů plánové, kresebné a fotografické dokumentace lidové architektury v ČR. Jeho smyslem je zpřístupnit tyto fondy širší veřejnosti. Toho se chce docílit vytvořením virtuálního skansenu, který bude obsahovat zejména 3D modely architektonických památek lidového stavitelství na území ČR. Dále bude vytvořena veřejná databáze mapových výstupů, plánů, fotografií a dalších dokumentů obohacená o množství metadat. Bude využívána především při vzdělávání, ale také prostřednictvím výstav a publikací.

Aplikace je zaměřená na interní správu objektů a nebude tedy přístupná veřejnosti. Měla by umět zobrazovat data z databáze převzaté z fondů Národního muzea. Ta bude dále rozšířena o uživateli poskytovaná data jako fotografie, modely, dokumenty a jejich metadata. Tato data už budou moci uživatelé přidávat, zobrazovat, mazat a editovat. Jeden z hlavních požadavků je také na oprávnění k objektům, tedy aby uživatelé mohli editovat pouze taková data, ke kterým mají udělené oprávnění. Úkolem projektového týmu je také zajistit jednotný vývoj a nasazení aplikace společně s databází.

V první kapitole práce se diskutuje o možných technologických řešeních práce jako je volba vývojového prostředí aplikace nebo databázového systému. Na to navazují technologie, použité jak při vývoji aplikace, tak pro úpravu databázové struktury nebo automatického nasazení aplikace. Dále je popsáno navázání na rozpracovanou aplikaci a použité databáze. Poslední kapitola se zabývá nejen samotným vývojem aplikace, ale také normalizací databáze a použitím softwaru Docker, pro vytvoření jednotného prostředí pro vývoj aplikace.

1 Rešerše

1.1 Volba metody pro tvorbu webové aplikace

Při volbě, jakou metodu pro tvorbu aplikace použít, se nabízelo několik možností jak ji vytvořit. Aplikace by mohla být vytvořena například za použití skriptovacího jazyka PHP, Javascriptu nebo Python frameworku pro tvorbu webových aplikací.

Ze zadání projektového týmu ovšem vyplynulo, že by aplikace měla být napsána v jazyce Python, a to z důvodu udržitelnosti kódu. Volba tedy padla na Python frameworky.

1.2 Volba Python frameworku

Frameworkem je označována platforma, která se používá pro tvorbu aplikací. Je to soubor podpůrných programů a knihoven, které mají za cíl usnadnit jejich vývoj. Výběr vhodného frameworku byl vzhledem k jejich množství složitý. Mezi hlavní kritéria pro výběr frameworku se proto zařadila snadná práce s databází, implementace oprávnění k objektům a dostupnost výukových materiálů a komunitní podpora. Díky těmto kritériím se nakonec vybíralo ze dvou frameworků, a to Flask a Django.



Obrázek 1.1: Flask vs Django (https://miro.medium.com/max/1672/1*TbMXbstb039keN-glX2CqQ.jpeg)

Flask je open source webový framework napsaný v jazyce Python, klasifikovaný jako mikro z důvodu, že není potřeba žádných dodatečných knihoven ani jiných nástrojů. Do Flasku lze ovšem doinstalovat různá rozšíření, která v základní verzi chybí jako například Flask-Admin, což je administrátorské rozhraní pro správu uživatelů a objektů v databázi. Výhodou je jeho velká obliba v komunitě, kdy v roce 2020 měl druhé místo na github.com z webových frameworků, a díky tomu disponuje

spoustou materiálů a tutoriálů. Avšak velká nevýhoda Flasku pro vývoj naší aplikace je, že nedisponuje datovými modely. Pokud bychom v průběhu vývoje chtěli přidat sloupec do naší databáze, musíme to udělat ručně v databázi a poté ho přidat do třídy ve webové aplikaci. [1]

Django je nejpoužívanějším open source frameworkem. Jednou z jeho funkcí je, že disponuje objektově relačním mapováním. Jeho vlastností je tedy možnost generovat model z databáze, který se zde může upravovat a poté jednoduchými příkazy promítnout úpravy zpět do databáze. Další výhodou je implementované administrátorské rozhraní a možnost doinstalováním přídatných balíčků, poskytující jak grafické úpravy, tak přidané funkce. Jedním z nich je i balíček django-guardian zajišťující podporu oprávnění k objektům. [2]

Z těchto frameworků byl nakonec vybrán pro tvorbu aplikace framework Django. Ten oproti Flasku disponoval snadnou a rychlou prací s modelem databáze. Další výhodou je vývojáři implementované administrátorské rozhraní, které se nemusí doinstalovat pomocí přídatného balíčku. Z hlediska popularity a dostupnosti výukových materiálů jsou na tom oba frameworky podobně, kdy se oba řadí na první dvě místa výrazně před ostatní frameworky.

1.3 Volba databáze

Jedním z hlavních úkolů bylo vytvoření databáze, ze které se budou zobrazovat poskytnutá data z projektu Viskalia, rozšířená o další položky. Textová data zprostředkovaná fondem Národního muzea byla uložena v MariaDB databázi a obrazová data byla v souborovém systému. Příslušná databáze ovšem postrádá jakoukoli normalizaci a data obsahují spoustu duplicit. Nabízí se proto otázka, zda provést normalizaci dat a opět použít relační databázi, nebo nějakou z nerelačních databází. Asi zásadní problém s NoSQL databází, který by mohl nastat je v komunikaci s frameworkem pro vývoj webových aplikací. Ty v zásadě nemají problém komunikovat s jakýmkoliv typem relační databáze, ovšem často nemají engine pro databáze typu NoSQL. Byla tedy zvolena relační databáze a to MariaDB, z důvodu poskytnutí dat právě z tohoto databázového systému. [2] [3]

1.4 Automatizace nasazení aplikace

Jedním z hlavních úkolů byla také automatizace nasazení aplikace a zajištění bezproblémového chodu na všech zařízeních. Při této otázce se projektový tým nejprve musel rozhodnout, na jakém operačním systému se bude aplikace vyvíjet a následně bude provozována. Po kratší úvaze se zvolil systém Linux, který je vhodnější zejména pro vývoj aplikací v jazyce Python. Zde se tedy hledal vhodný software pro kontejnerizaci naší vyvíjené aplikace. Hlavními důvody pro jejich použití je, že obsahují kompletní prostředí pro vývoj a chod aplikace jako jsou knihovny nebo konfigurační soubory. Aplikace a všechny její komponenty většinou běží v několika kontejnerech, které spolu dokáží komunikovat. Jejich chod je od zbytku systému oddělen. Zde se nabízejí dvě varianty, a to jsou LXC (LinuX Containers) anebo Docker, což jsou jedny z nejpoužívanějších aplikací. Hlavní rozdíl mezi těmito aplikacemi je, že kontejnery LXC obsahují svůj vlastní operační systém a působí tedy spíše jako virtuální počítač. Docker oproti tomu žádné takové prostředí nemá a běží pouze separovaně na operačním systému uživatele. Dalším hlavním rozdílem je větší propracovanost Dockeru, které umožňuje více nastavení kontejnerů a rozdělení aplikace do více kontejnerů. Oproti LXC může být Docker použit i na jiném operačním systému než je Linux. Z těchto důvodů byl pro kontejnerizaci zvolen Docker. [4]

1.5 Normalizace databáze

Normalizace databáze je reorganizace dat podle normalizačních standardů. Používá se pro snížení nadbytečných dat v databázi, například při duplicitách ve sloupcích. Normalizačních forem je pět, ale zpravidla se používají pouze první tři stupně. Jednotlivé stupně na sebe navzájem navazují, pokud budeme tedy chtít uplatnit třetí normalizační stupeň, musíme nejprve provést první a druhý.

První stupeň je v zásadě použitím selského rozumu. Jedná se o odstranění duplicitních sloupců a sloupců, jež jsou kombinací jiných sloupců.

Druhý stupeň nám říká, že duplicitní hodnoty by měly být z tabulky odstraněny a přesunuty do nové tabulky, která je propojená cizím klíčem. Z tabulky se tedy odstraní všechny hodnoty, které platí pro více záznamů.

Třetí stupeň udává, že by tabulka neměla obsahovat hodnoty, které jsou na sobě závislé.

Ve čtvrtém stupni by záznamy neměly obsahovat více nezávislých hodnot jedné podmnožiny. Takovéto hodnoty by opět měly být převedeny do samostatné tabulky a spojeny cizím klíčem.

Pátý stupeň platí, pokud není možné tabulku dále dělit bez ztráty dat. Po rozdělení by došlo ke ztrátě vztahů, které mezi daty platí. [5]

1.6 Objektově relační mapování

Objektově relační mapování (ORM) zajišťuje konverzi dat mezi relační databází a objektově orientovaným programovacím jazykem. Umožňuje konverzi SQL tabulek do tříd a usnadňuje tak práci s psaním SQL dotazů. Pro tuto konverzi se často používají externí programy, které umožňují tyto konverze mezi řadou databázových systémů. Jedním z nich je například SQLAlchemy, které umožňuje tuto konverzi pro programovací jazyk Python a relační databáze SQLite, Postgresql, MySQL a další. Některé aplikace včetně Django si ovšem tuto konverzi zprostředkovávají sami za pomoci vlastního ORM. [6] [7]

2 Použité technologie

2.1 Django



Obrázek 2.1: Django Logo (<https://www.djangoproject.com/community/logos/>)

Django je open source prostředí, pro tvorbu webových aplikací. Django je napsané v programovacím jazyku Python a pracuje na bázi model-template-view. Model v tomto případě reprezentuje část, která je schopna komunikovat a pracovat s databází. Templates určují obsah zobrazení a jsou realizovány pomocí HTML. View zobrazuje daný obsah koncovým uživatelům.

Django bylo vytvořeno v roce 2005 americkou společností The World Company. Jeho první verze byla označena 0.90 a vyšla 16. listopadu 2005. Vývojáři pokračovali v jeho zdokonalování, kdy bylo během následujících pár let provedeno několik aktualizací. V roce 2008 byla vydána verze 1.0. Vývoj se nezastavil a po několika aktualizacích byla v roce 2013 vydána verze 1.5, která zajišťovala také podporu pro Python 3, dříve pouze Python 2. Další velkou aktualizací bylo vydání nové verze Django 2.0 na konci roku 2017. Ta zajišťovala plnou podporu pro Python 3 a nadála už není možné pracovat s verzí Python 2. Aktuální verze Djanga je 3.1, s tím že se plánuje v blízké době vydat verzi 3.2 po které by měl následovat další velká aktualizace v podobě Djanga 4.0, která je plánována na začátek roku 2022. [2]

2.1.1 Django - instalace a inicializace projektu

Django je framework pracující v jazyce Python, pro jehož aktuální verzi 3.1 je potřeba mít nainstalován verzi Python 3.6 nebo vyšší. Nejjednodušší je pomocí příkazové řádky spustit příkaz `pip install django`, čímž by se měla nainstalovat nejaktuálnější verze Django. Po instalaci lze pro zobrazení aktuální verze použít příkaz `django-admin -version`. Po instalaci se opět pomocí příkazové řádky provede vytvoření projektu v aktuální složce příkazem `django-admin startproject [název projektu]`. [8] Tímto se vytvoří nový adresář který vypadá následovně:

- [Název projektu]/
 - manage.py
 - [Název projektu]/
 - * `__init__.py`
 - * `setting.py`
 - * `urls.py`
 - * `asgi.py`
 - * `wsgi.py`

`__init__.py` určuje, aby s adresářem bylo zacházeno jako s celkem.

`setting.py` zde se definují všechna nastavení Django projektu.

`urls.py` v projektovém adresáři definuje seznam URL adres na úrovni projektu. Při založení je zde obsažena administrátorská stránka.

`wsgi.py` (Web Server Gateway Interface) slouží pro propojení webového serveru s frameworkem Django, popřípadě jinými webovými aplikacemi.

`asgi.py` (Asynchronous Server Gateway Interface) byl do Django přidán ve verzi 3.0 a je nástupce staršího WSGI. Výhodou WSGI oproti ASGI je, že dokáže pracovat se složitějšími protokoly jako jsou WebSocket, IoT protokoly, HTTP2 a další. Umožňuje komunikovat přes více linek a tedy posílat a přijímat více dat najednou.

`manage.py` je automaticky vygenerován v novém adresáři a je přítomen v každém Django projektu. Je to nástroj pro spuštění specifických příkazů například pro spuštění aplikace nebo vytvoření aplikací.

Pomocí nástroje `manage.py` se v Django spouští řada důležitých příkazů. Zde je výčet těch nejdůležitějších a jejich popis. [9]

- `python manage.py runserver` - tento příkaz spouští aplikaci na lokálním serveru. Standardně je to na adrese `127.0.0.1:8000`. Aplikace je po spuštění přístupná pouze z daného zařízení a není možné se k ní připojit ani v rámci lokální sítě. Pokud je při spuštění serveru provedena nějaká změna v kódu, framework projde celý projekt pro nalezení případné chyby a změna se automaticky zobrazí ve výstupu.
- `python manage.py makemigrations [název aplikace]` - tento příkaz provede porovnání modelu s databází. Pokud jsou nalezeny změny, vytvoří se nový soubor se stávajícími migracemi. Ten obsahuje SQL příkazy, jak by se měla vytvořit databáze podle modelu definovaném v aplikaci. Příkaz se definuje zadáním jména aplikace, potažmo aplikací. Lze tedy vytvořit migrace pro jednu nebo více aplikací zadáním jejich názvů.
- `python manage.py migrate` - spuštěním tohoto příkazu se synchronizuje databáze s modelem podle migračního souboru, který je generovaný z příkazu `python manage.py makemigrations [název aplikace]`.
- `python manage.py inspectdb` - příkaz vytvoří schéma připojené databáze a uloží ho do paměti. Vytvoření `model.py` souboru se pak provede spuštěním příkazu `python manage.py inspectdb > models.py`.

2.1.2 Django - tvorba aplikace

Po založení projektu je potřeba vytvořit aplikaci. Počet použitých aplikací v projektu není omezen a jejich výhodou je, že je lze opětovně použít v dalších projektech. Aplikace se vytvoří příkazem `python manage.py startapp [název aplikace]`. [8] Schéma je následující:

- [Název aplikace]/

```

– __init__.py
– admin.py
– apps.py
– models.py
– urls.py
– [migrations]/
  * __init__.py
  
```

`__init__.py` má stejný význam jako v adresáři projektu.

admin.py slouží k registraci modelů do administrátorské aplikace a k jejich úpravě.

apps.py je konfigurační soubor, který obsahuje metadata o aplikaci.

models.py obsahuje model databáze použitý v aplikaci.

urls.py obsahuje všechny url adresy aplikace.

tests.py obsahuje testovací metody, které se spustí při testování funkčnosti aplikace.

migrations ve složce migrations se nacházejí všechny migrační soubory, vytvořené při spuštění příkazu `python manage.py makemigrations [název aplikace]`.

2.1.3 Django - práce s databází

Další z věcí, kterou Django nabízí je jednoduchá a rychlá práce s databází. Při práci s databází se používá soubor `models.py`, který je při vytvoření projektu prázdný. Django standardně používá databázi SQLite, která se automaticky vytvoří v adresáři projektu. V souboru `settings.py` lze ovšem nastavit připojení k vlastní, již existující databázi. Při tvorbě databáze se do `models.py` tedy vytváří modely jednotlivých tabulek, které poté pomocí příkazů `python manage.py makemigrations [název aplikace]` a `python manage.py migrate` vytvoří nebo upraví tabulky v databázi. S databází jako takovou tedy vůbec nemusí pracovat. Při připojování k již existující databázi lze příkazem `python manage.py inspectdb` vygenerovat model stávající databáze. Django oficiálně podporuje databáze PostgreSQL, MariaDB, MySQL, Oracle a SQLite. Jsou zde ovšem i komunitně vytvořené balíčky pro podporu NoSQL databáze MongoDB. [2]

Model obsahuje všechny informace o tabulkách a jejich sloupcích. Jednotlivé tabulky jsou v modelu vytvořeny jako třídy `class jmeno_tabulky(models.Model)`. Každá třída pak obsahuje názvy jednotlivých sloupců a jejich datové typy. V závorce jsou pak obsaženy dodatečné informace jako například primární klíč, maximální délka proměnné atd `first_name=models.CharField(max_length=30)`. Jednotlivé třídy dále mohou obsahovat další třídy, například `meta`, která obsahuje dodatečné informace o tabulce.

```

from django.db import models
from django.contrib.auth import models as auth_models

# Create your models here.
class squad(models.Model):
    squad_name = models.CharField(max_length=30)
    base = models.CharField(max_length=30, blank=True)

class heroes(models.Model):
    id = models.AutoField(primary_key=True)
    nickname = models.CharField(max_length=20)
    powers = models.CharField(max_length=50, null=True, blank=True)
    can_change = models.BooleanField(default=False)
    managers = models.ManyToManyField(auth_models.User, null=True, blank=True)

    def races(self):
        return ', '.join(race.hero_race for race in self.race_set.all())

class race(models.Model):
    hero_race = models.CharField(max_length=20)
    post = models.ForeignKey(heroes, default=None, on_delete=models.CASCADE)

    def __str__(self):
        return f"{self.hero_race}"

```

Obrázek 2.2: Ukázka `models.py`

2.1.4 Django - přídatné balíčky

Packages, neboli balíčky jsou nástroje pro usnadnění práce s Djangem. Tyto balíčky mají mnoho funkcí, a to od změny vzhledu stránky, až po implementaci nových funkcí do administrátorské aplikace. Většina balíčků je tvořena komunitou a každý uživatel si také může vytvořit svůj vlastní. Pro nalezení vhodného balíčku slouží například stránka <https://djangopackages.org>, kde lze najít všechna rozšíření. Ty jsou zpravidla uloženy na stránce github.com, kde je detailně popsána jejich dokumentace. Jejich implementace je opravdu jednoduchá, pomocí pip se nainstaluje příslušný balíček (např. `pip install django-rest`). Dále se v projektu v souboru `settings.py` přidá rozšíření do `INSTALLED_APPS` a provede se migrace pro vytvoření tabulek v databázi. [10]

2.1.5 Django - administrátorské rozhraní

Jeden z nejdůležitějších komponentů Djanga je administrátorské rozhraní, které dovoluje oprávněným uživatelům pracovat s daty uloženými v databázi. Pomocí registrovaných modelů může uživatel s oprávněním prohlížet, přidávat, upravovat a mazat data v jednotlivých modelech. Administrátorská aplikace je od té uživatelské oddělena a je možné se do ní dostat přidáním `/admin` za webovou adresu. Pro přihlášení musí být uživatel registrován jako *staff* a poté mu musí být udělena práva k jednotlivým modelům. V aplikaci je možné nastavit práva jednotlivcům, popřípadě skupinám, do kterých se poté uživatelé přidávají.

Registrace modelů a jejich následná úprava se provádí v souboru `admin.py`. Nejprve se modely musí importovat a poté registrovat pro zobrazení v administrátorském rozhraní. Pokud je potřeba tabulky modifikovat, například zobrazit jen některá data, je třeba vytvořit ke každému modelu třídu, která se následně upravuje. Pro tento případ Django disponuje velkou škálou funkcí pro jednoduchou úpravu zobrazení dat. [11]

2.2 Docker

Docker je open source software používaný pro jednotný vývoj aplikací, které běží v izolovaných kontejnerech. Je primárně využíván na operačním systému Linux, ale je možné ho zprovoznit také na Windows nebo Mac OS X. Výhodou je také, že může běžet jak lokálně, tak na cloudovém uložení. Jeho užití spočívá ve vytvoření obrazů (image), ze kterých se následně v kontejnerech spustí aplikace ve stejné formě, v jaké byl vytvořen její obraz. Díky tomu je poté zajištěn bezproblémový chod na jakémkoliv jiném zařízení se stejným operačním systémem. Image pro Linux může běžet pouze na systémech s Linux a stejně tak je to pro Windows. Aplikace spuštěná v Dockeru je izolovaná, a nezasahuje do ostatních aplikací. Je zde také snadná možnost testovat nové verze knihoven a aplikací.

Při použití Dockeru je nutné nejprve vytvořit obraz, který obsahuje všechny aplikace a jejich nastavení potřebné pro bezproblémový chod. Z tohoto obrazu se poté vytvoří kontejner, který obsahuje také svou vlastní vrstvu, kde jsou uloženy všechny změny, které uživatel provedl. [12]

2.3 Git

Git je open source verzovací systém, který se hojně používá při vývoji aplikací. Jeho použití je možné na řadě platform a ovládání se provádí přes příkazový řádek nebo GUI. Git funguje na principu repozitářů, ve kterých jsou uloženy projekty. Repozitář obsahuje celý projekt a jeho kompletní historii provedených změn. Každý člen týmu je tedy rovněž i zálohou. Při použití Gitu se využívají webové služby jako například GitHub nebo GitLab, kde jsou repozitáře zálohovány. Zároveň tu je mnoho užitečných nastavení jako veřejné nebo soukromé repozitáře, nastavení práv uživatelů a spousta dalších. [13]

2.4 MariaDB

MariaDB je komunitně vyvíjeným databázovým systémem odvozeným od MySQL licencovaný jako svobodný software GNU. První verze MariaDB vyšla v roce 2009 a měla číslo 5.1, to odpovídalo aktuální verzi MySQL, poslední verze této řady byla verze 5.5. Další řada nesla označení 10 a lišila se tím, že už nebyla 100% kompatibilní s MySQL. Aktuálně nejnovější verze je 10.5, která vyšla na konci roku 2019. MariaDB podporuje řadu operačních systémů včetně těch nejrozšířenějších jako Windows, Linux a MacOS. Vzhledem k tomu, že se vycházelo z MySQL, má MariaDB stejnou strukturu a indexování. Oproti MySQL má ovšem daleko rychlejší vývoj, což je způsobeno i tím, že je celý projekt open source. [3]

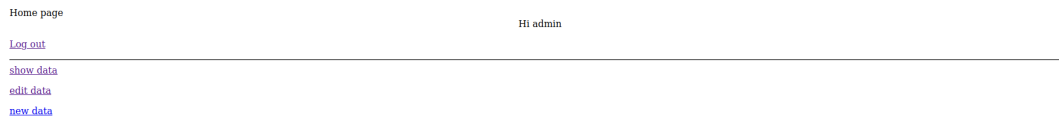
2.5 phpMyAdmin

Je to open source nástroj pro správu MySQL a MariaDB databáze pomocí webového rozhraní. Je napsaný v jazyce PHP a je k dispozici ve více jak 70 jazycích včetně češtiny. Toto rozhraní umožňuje provádět SQL příkazy jako jsou CREATE, DROP, ALTER table a spravovat obsah tabulek. Dále je zde možný například import dat z CSV souborů, export dat do nejrůznějších formátů nebo správa uživatelů. K přístupu do rozhraní phpMyAdmin je zapotřebí webový prohlížeč s podporou cookies a JavaScriptu. [14] [15]

3 Seznámení s aplikací a databází

3.1 Navázání na projekt FGIS

Tato diplomová práce navazuje na započatý projekt z předmětu Free Software GIS. Na projektu se také podílel autor práce. Projekt měl za úkol vytvořit jednoduchou webovou aplikaci, kdy přihlášení uživatelé mohli zobrazovat, vytvářet, editovat a mazat záznamy z databáze. Databáze pro tento projekt byla pouze cvičná a obsahovala pouze několik tabulek s textovými poli. Projekt ovšem nebyl zcela dokončen, protože se nepovedlo vytvořit všechny funkce tak, aby správně fungovaly. Data do databáze šla přidávat, zobrazovat je a mazat, ale při editaci se po uložení nepřepsala aktuální data v databázi.



Obrázek 3.1: Náhled aplikace vytvořené v projektu FGIS

3.2 Zkušební databáze

Na začátku projektu byla používána pouze zkušební databáze, jejíž použití bylo z důvodu neznalosti přesné struktury finální databáze. Zkušební databáze sloužila na začátku projektu pro inicializaci aplikace a vytvoření základních funkcionalit. Databáze obsahovala pouze tabulku s textovými daty.

3.3 Centrální databáze

Databáze fondů Národního muzea obsahuje objekty, které se budou ve vyvíjené aplikaci zobrazovat a budou rozšířena o další data. Všechna data v této databázi jsou obsažena v jedné tabulce. Část sloupců v této tabulce obsahuje informace o objektu a část o obrázku, který je k němu přidělen. Jednotlivé objekty ovšem disponují více obrázky, kdy se pro každý obrázek vytváří nový databázový záznam. Zde dochází k duplikování velké části dat, kdy se sloupce s informacemi o objektu kopírují a mění se pouze část s obrázky a jejich metadaty. Databáze je tedy v nenormalizované formě.

OID	ID	Instituce	Fond	Inventarní_cislo	Signatura	Inkrement	Karton_ulozeni	Lokalita	Puv_nazev	CP
1	99	EÚ AV ČR	Zaměřovací akce ČAVU	0108	0	0	EÚ AV ČR Praha, archiv, mapník I/1	Smilkov		14
1	100	EÚ AV ČR	Zaměřovací akce ČAVU	0109	0	0	EÚ AV ČR Praha, archiv, mapník I/1	Smilkov		14
1	101	EÚ AV ČR	Zaměřovací akce ČAVU	0110	0	0	EÚ AV ČR Praha, archiv, mapník I/1	Smilkov		14
1	102	EÚ AV ČR	Zaměřovací akce ČAVU	0111	0	0	EÚ AV ČR Praha, archiv, mapník I/1	Smilkov		14
1	103	EÚ AV ČR	Zaměřovací akce ČAVU	0112	0	0	EÚ AV ČR Praha, archiv, mapník I/1	Smilkov		14
1	104	EÚ AV ČR	Zaměřovací akce ČAVU	0113	0	0	EÚ AV ČR Praha, archiv, mapník I/1	Smilkov		14
1	105	EÚ AV ČR	Zaměřovací akce ČAVU	0114	0	0	EÚ AV ČR Praha, archiv, mapník I/1	Smilkov		14

Obrázek 3.2: Náhled dat centrální z databáze s viditelnými duplicitami

4 Tvorba aplikace

4.1 Vytvoření základní aplikace

Aplikace, která byla vytvořena v projektu FGIS nebyla vytvořena správně, obsahovala zbytečně složitý a chybný kód. Začalo se tedy od začátku. Nejprve byl vytvořen projekt příkazem `django-admin startproject mysite`. V projektu následně byla vytvořena aplikace pomocí `python manage.py startapp`. Aplikace byla v `settings.py` přidána do `INSTALLED_APPS`. Dále se nastavilo připojení k databázi. V `settings.py` se tedy v `DATABASES` změnil engine na MySQL, protože Django v základu používá SQLite. Změnilo se ještě `NAME`, `USER`, `PASSWORD`, `HOST` A `PORT`, protože databáze byla provozována na školním serveru `geo102.fsv.cvut.cz`. Po připojení se musel vygenerovat model databáze příkazem `inspectdb`. Dále byla provedena migrace, aby se v databázi vytvořily Django tabulky. Posledním krokem bylo vytvoření superuživatele, který se bude moci přihlásit do uživatelské i administrátorské aplikace. To bylo realizováno příkazem `python manage.py createsuperuser`.

Na stránce GitLab.com byl vytvořen nový repozitář, do kterého se celý projekt přesunul a nadále v něm byl průběžně zálohován. Dále se tam vytvářely tzv. *issues*, kde se plánovaly nadcházející úkoly při tvorbě aplikace.

4.2 Tvorba aplikace v uživatelském prostředí

Při vytváření uživatelské aplikace bylo postupováno dle příkladu v [16]. Prvním větším úkolem bylo vytvořit aplikaci, která bude zobrazovat data a po přihlášení je bude moci uživatel také přidávat, mazat a editovat. Nejprve se tedy do adresáře aplikace přidal soubor `urls.py`, který se následně připojil k `urls.py` v projektovém adresáři. Tento způsob je výhodný při použití více aplikací, kdy si každá aplikace uchovává své URL adresy a projekt je tak přehlednější. Pokračovalo se vytvořením pohledů založených na třídách, které se v Pythonu používají pro nahrazení pohledů jako funkcí. Použity jsou základní pohledy jako `ListView` pro zobrazení obsahu tabulky nebo `CreateView` pro uložení dat do databáze. V aplikaci v soubory `views.py` jsou tedy vytvořeny třídy pohledů pro každou URL adresu. Definice těchto tříd vypadá následovně *class* “název třídy“(Listview): dále se u třídy definuje model,

který se má zobrazit a `template_name` neboli název šablony, která se použije při zobrazení.

```

from django.views.generic import ListView, DetailView
from django.views.generic.edit import CreateView, UpdateView, DeleteView
from django.urls import reverse_lazy
from .models import Cvut, CvutImages

class HomeView(ListView):
    model = Cvut
    template_name = 'home.html'

class ShowData(ListView):
    model = Cvut
    template_name = 'show_data.html'

class EditData(ListView):
    model = Cvut
    template_name = 'data.html'

class DataDetail(DetailView):
    model = Cvut
    template_name = 'data_detail.html'

```

Obrázek 4.1: Náhled `views.py` a jeho tříd

V souboru `urls.py` jsou propojeny URL adresy, pohledy a HTML šablony, na které se odkazují. Do `urls.py` musí být zároveň importovány všechny použité pohledy. Dále je potřeba vložit cestu příkazem `import path`. Poté se sestaví `urlpatterns`, což je seznam, který obsahuje jednotlivé adresy. Ty jsou v něm uvedeny ve funkci `path`, která má dva povinné argumenty, `route` a `view` a dva nepovinné, `name` a `kwargs`. `Route` uvádí adresu, pod jakou se bude stránka načítat a `view` odkazuje na importovaný pohled. Zde byla použita funkce `as_view()`, která se používá, pokud je pohled typu třída. Jméno je uvedeno jako název HTML šablony.

```

from django.urls import path

from .views import HomeView, ShowData, EditData, DataDetail, EditDetail, NewData, DeleteData

urlpatterns = [
    path('data/<int:pk>/delete/', DeleteData.as_view(), name='data_delete'),
    path('data/<int:pk>/edit/', EditDetail.as_view(), name='edit_detail'),
    path('data/<int:pk>/', DataDetail.as_view(), name='data_detail'),
    path('data/new/', NewData.as_view(), name='data_new'),
    path('data/', EditData.as_view(), name='data'),
    path('show/', ShowData.as_view(), name='show_data'),
    path('', HomeView.as_view(), name='home'),
]

```

Obrázek 4.2: Náhled urls.py v adresáři aplikaci

Pro vytvoření HTML šablon se vytvořila složka *templates*, kde jsou uloženy jednotlivé HTML soubory. Tato složka se v *settings.py* nastaví jako výchozí pro jejich zobrazení. Poté se už mohou vytvářet jednotlivé šablony. Šablony jsou vždy vytvořeny s podmínkou *if user.is_authenticated*, tedy pokud je uživatel přihlášen, zobrazí se mu na stránce možnosti data editovat, vytvářet a mazat. S tímto je spojena ještě další změna v *settings.py*, kde je přidáno `LOGIN_REDIRECT_URL = 'home'` a `LOGOUT_REDIRECT_URL = 'home'` které uživatele po přihlášení a odhlášení přesměruje na domovskou stránku. Ve složce *templates* je ještě složka *registration*, kde je umístěn *login.html*. HTML soubory zobrazují data v základní formě tabulek a nejsou graficky upravována pomocí CSS (Cascading Style Sheets).

```

{% block content %}
{% if user.is_authenticated %}
<h1>Edit object
<hr><a href="{% url 'data' %}">edit data</a>
<form action="" method="post" enctype="multipart/form-data">
{% csrf_token %}
{{ form.as_p }}
<input type="submit" value="Update" />
</form>
{% else %}
<p>you are not logged in
{% endif %}
{% endblock content %}

```

Obrázek 4.3: Náhled HTML souboru edit detail

4.3 Přidání obrázků

Dalším úkolem bylo vyřešit přidávání obrázků k jednotlivým záznamům. Obrázky se neměly ukládat do databáze, ale do adresáře v aplikaci. Databáze poté měla obsahovat cestu k těmto souborům. K tomu byla vytvořena nová tabulka, který byla přes cizí klíč spojena s tabulkou se záznamy. Tabulka byla vytvořena v modelu, a poté pomocí migrací byla přenesena do databáze. Obsahuje tedy pole ID, což je primární klíč tabulky, Post, který je cizím klíčem k tabulce Cvut a sloupec Image, do kterého se zaznamenává cesta k danému souboru viz 4.13. Pro zobrazení obrázků ještě musel být doinstalován přídatný modul Pillow.

```
class CvutImages(models.Model):
    ... id = models.AutoField(db_column='ID', primary_key=True)
    ... post = models.ForeignKey(Cvut, default=None, on_delete=models.CASCADE)
    ... images = models.ImageField(upload_to='images/', blank=True, null=True)
```

Obrázek 4.4: Náhled tabulky CvutImages pro ukládání obrázků

V nastavení se dále určí kořenový adresář pro nahrávané soubory MEDIA_ROOT = os.path.join(BASE_DIR, 'media') a MEDIA_URL = '/media/'. A v poslední řadě se musí nastavit v projektovém *urls.py* media soubory, které Django samo neumí zobrazovat. Zde se provede import settings a static a k urlpatterns se připojí funkce static.

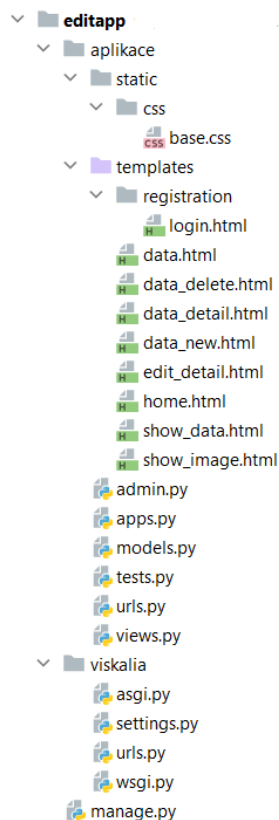
```
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('accounts/', include('django.contrib.auth.urls')),
    path('', include('aplikace.urls')),
] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Obrázek 4.5: Náhled projektového urls.py

4.4 Statické soubory

Pro řešení vizuální stránky webu jsou používány CSS soubory. Ve složce aplikace se tedy vytvoří adresář *static* a tomu se v *settings.py* určí jeho nastavení pro statické soubory příkazy `STATIC_URL = '/static/'` a `STATICFILES_DIRS = [os.path.join(BASE_DIR, 'static')]`. Ve složce *static* je vytvořen adresář *css*, kde se nachází základní CSS soubor *base.css*. Ten je ovšem prázdný a není připojen k žádnému HTML souboru.



Obrázek 4.6: Náhled projektového adresáře

4.5 Nastavení jazyka a času

Dalším nastavením bylo použití našeho středoevropského času a českého jazyka. Velice jednoduché nastavení, kdy v *settings.py* je po vytvoření aplikace nastaven jazyk na angličtinu a čas na UTC. Pro změnu se tedy přepíše `LANGUAGE_CODE = 'en-us'` na `LANGUAGE_CODE = 'cs'` a `TIME_ZONE = 'UTC'` na `TIME_ZONE = 'CET'`.

4.6 Administrátorské prostředí

Po vytvoření základní uživatelské aplikace a úvahách o její funkci jsme dospěli k rozhodnutí, že nám bude stačit administrátorské rozhraní, ve kterém se budou provádět modifikace podle daných požadavků. Při jeho tvorbě a modifikaci byla jako podklad použita oficiální dokumentace Django (*docs.djangoproject.com*) a [17]. Administrátorské prostředí se upravuje v adresáři aplikace v souboru *admin.py* a lze se do něj dostat zadáním */admin* za URL adresu webové stránky. Začalo se tedy importem a zaregistrováním modelů do rozhraní a vytvořením jejich tříd. Pro model *CvutImages* se nejprve vytvořila třída *PostImageAdmin*, ve které je použit *StackedInline* model. Ten se používá pro vnořené třídy, tedy pokud je chceme zobrazit na jedné stránce společně s jinou třídou. V tomto případě tedy chceme zobrazit obrázky, které přísluší jednotlivým záznamům v třídě *Cvut*. *@admin.register* slouží k zobrazení třídy v administrátorském rozhraní, podmíněno vytvořením třídy tohoto modelu, ve které se poté provádí modifikace zobrazení. Ta je zde rozšířena o třídu *PostImageAdmin*. Dále je zaregistrován model *CvutImages* a vytvořena jeho třída pro jeho zobrazení.

```

from django.contrib import admin

from .models import Cvut, CvutImages

class PostImageAdmin(admin.StackedInline):
    model = CvutImages

@admin.register(Cvut)
class PostAdmin(admin.ModelAdmin):
    inlines = [PostImageAdmin]

    class Meta:
        model = Cvut

@admin.register(CvutImages)
class PostImageAdmin(admin.ModelAdmin):
    pass

```

Obrázek 4.7: Registrace modelů v *admin.py*

4.7 Oprávnění k objektům

Dalším úkolem bylo zajištění oprávnění k objektům (per object permissions), tedy že uživatel nebude moci editovat celou tabulku, ale jen objekty, ke kterým bude mít udělená práva. Tuto možnost zajišťuje přídatný balíček *django-guardian*, který ovšem po jeho instalaci a implementaci nefungoval správně a i po nastavení práv uživatel nemohl daný objekt editovat. Vymyslel se tedy systém za použití autentifikačního systému, kdy se do tabulky *Cvut* přidá nová proměnná *Managers*, ve které se budou moci vybírat registrovaní uživatelé. Tato proměnná vytváří v databázi novou tabulku, která obsahuje ID, ID uživatele a ID objektu v tabulce. Ve vytvořených třídách v *admin.py* je poté možnost pomocí funkcí `has_view_permission()`, `has_add_permission()`, `has_change_permission()` a `has_delete_permission()` modifikovat práva k tabulkám. Zde se tedy změnilo nastavení funkcí tak, aby tabulky mohl editovat superuživatel, uživatel, který má udělená práva editovat celou tabulku nebo uživatel, který je přidán v tabulce *Managers*.

```
tmp5 = models.TextField(db_column='TMP5', blank=True, null=True)
fond2 = models.TextField(db_column='Fond2', blank=True, null=True)
gps_sirka = models.FloatField(db_column='GPS_sirka', blank=True, null=True)
gps_delka = models.FloatField(db_column='GPS_delka', blank=True, null=True)
gps_presne = models.IntegerField(db_column='GPS_presne', blank=True, null=True)
managers = models.ManyToManyField(auth_models.User, blank=True)
```

Obrázek 4.8: Vytvoření managers v modelu aplikace

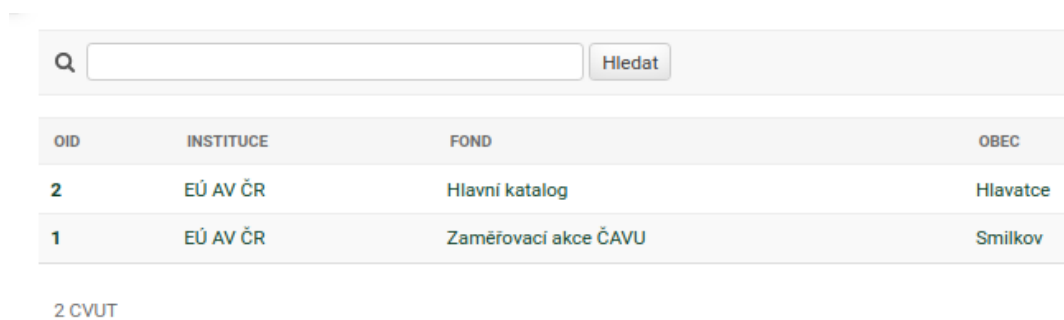
```
def has_change_permission(self, request, obj=None):
    if request.user.is_superuser:
        return True
    if obj is None:
        return True
    if request.user in obj.managers.all():
        return True
    return False

def has_add_permission(self, request, obj=None):
    if request.user.is_superuser:
        return True
    if obj is None:
        return True
    if request.user in obj.managers.all():
        return True
    return False
```

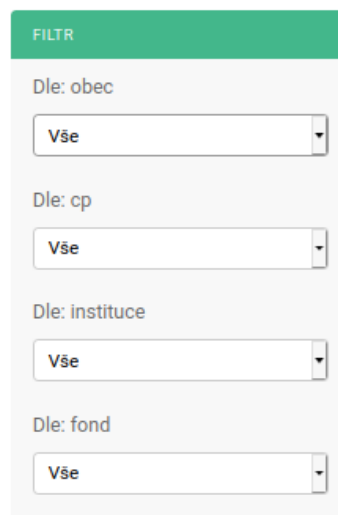
Obrázek 4.9: Úprava práv k objektům

4.8 Nastavení zobrazení dat a jejich vyhledávání

Django disponuje řadou funkcí pro nastavení zobrazení dat v administrátorském prostředí. Nejprve se nastavovalo zobrazení dat na stránce tabulky. Zde se měly zobrazovat pouze základní informace o objektu jako je ID, fond, okres, obec. To se nastavuje pomocí proměnné *list_display* a *list_display_links* slouží pro určení položek, díky kterým se po kliknutí dostanete na stránku obsahující informace o daném objektu. Na této stránce se zobrazují všechny určené položky definované v listu *fields*. Dále byly nastaveny filtry pomocí *list_filer* a vyhledávání *search_fields*.



Obrázek 4.10: Zobrazení dat a vyhledávání



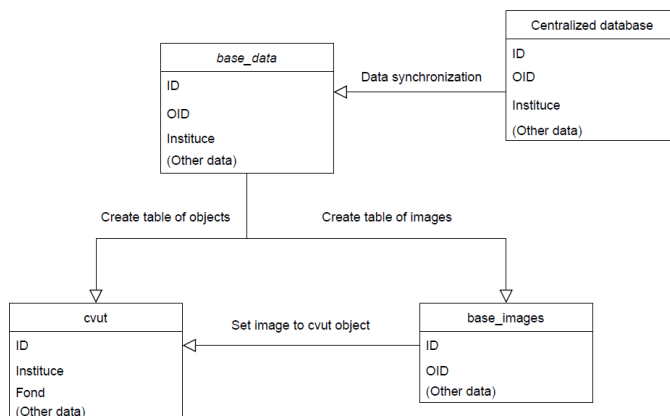
Obrázek 4.11: Vyhledávání podle filtrů

4.9 Použití balíčku `admin_interface`

Při hledání jednoduché a efektivní cesty pro úpravu administrátorského prostředí byl vybrán balíček `admin_interface`. Umožňuje superuživateli po přihlášení do aplikace možnost editovat vizuální stránku aplikace jako změnu loga, nadpisu nebo barvy textu a pozadí. Instalace balíčku viz kapitola 2.1.4 Django - balíčky. Pro správnou funkci je nutno `INSTALLED_APPS` v `settings.py` rozšířit nejen o `admin_interface` ale také o `colorfield`.

4.10 Tvorba databáze

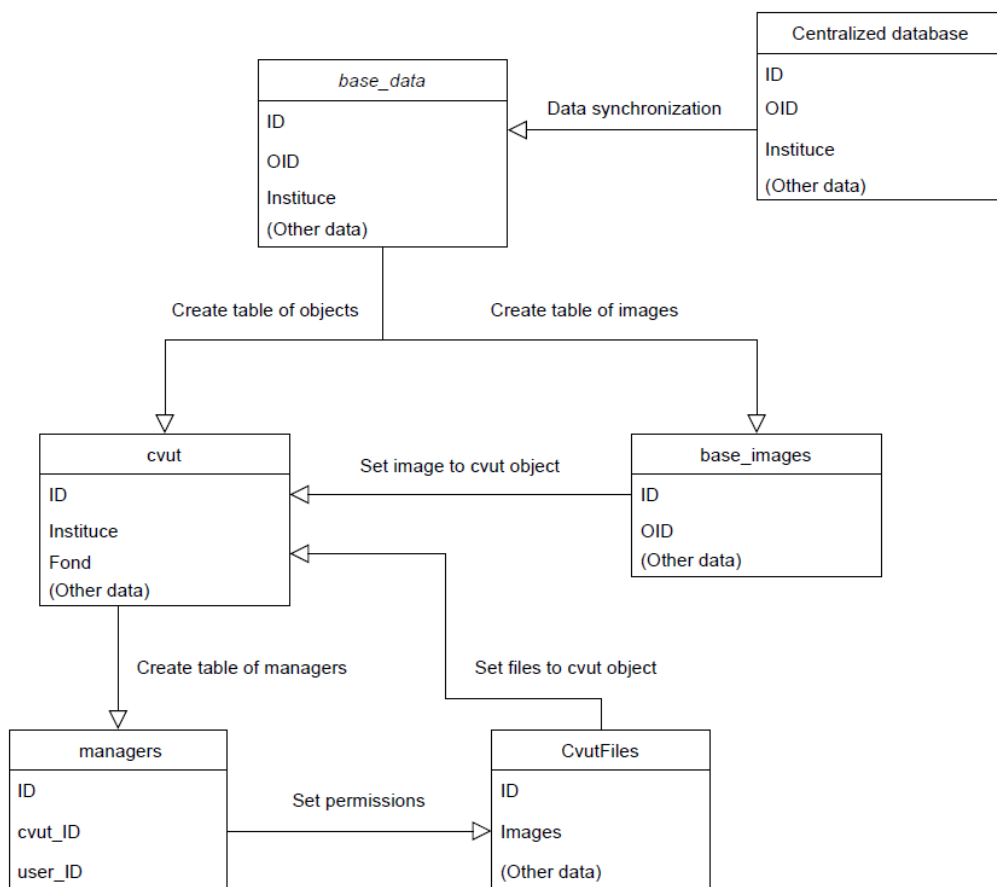
Po vytvoření základních funkcionalit aplikace byla projektovému týmu zprostředkována databáze s daty, která se mají zobrazovat. Forma poskytnutých dat byla specifikována v kapitole [3.3]. Django nedisponuje jednoduchým řešením, jak redukovat duplicitní záznamy a přiřadit jednotlivé obrázky k jednomu záznamu. Úkolem projektového týmu tedy bylo vymyslet řešení, které by tento problém odstranilo. Jako nejlepší řešení se ukázalo z původní tabulky (`Base_Data`) exportovat data se sloupci, které příslušely k jednotlivým objektům a vložit je do nové tabulky. Tímto vznikla samostatná tabulka všech objektů (`Cvut`). Dále se z původní tabulky vybraly sloupce, kde byly umístěny informace o uložených fotografiích a byly opět vloženy do samostatné tabulky (`Base_Images`). Tato tabulka byla pomocí cizího klíče propojena s tabulkou `Cvut`, kde se fotografie odkazovaly na jednotlivé objekty. Data se tedy dostala do normalizované podoby podle třetího stupně.



Obrázek 4.12: Struktura normalizované databáze

4.11 Doplnění výběrové databáze

Tato databáze už obsahuje data z centrální databáze, které se pomocí skriptu upravily dle kapitoly [4.10]. Po konzultaci projektového týmu se dospělo k závěru, že se k jednotlivým objektům budou přidávat nejen obrázkové soubory, ale také modely a jiné dokumenty. Vytvořila se tedy tabulka CvutFiles, která odpovídala tabulce z kapitoly [4.3], ovšem s tím rozdílem že místo ImageFile byl použit FileField a byla doplněna o textové informace jako popis nebo typ dokumenty. Tabulka Cvut se opět rozšířila o sloupec Managers viz kapitola [4.7].



Obrázek 4.13: Struktura výběrové databáze

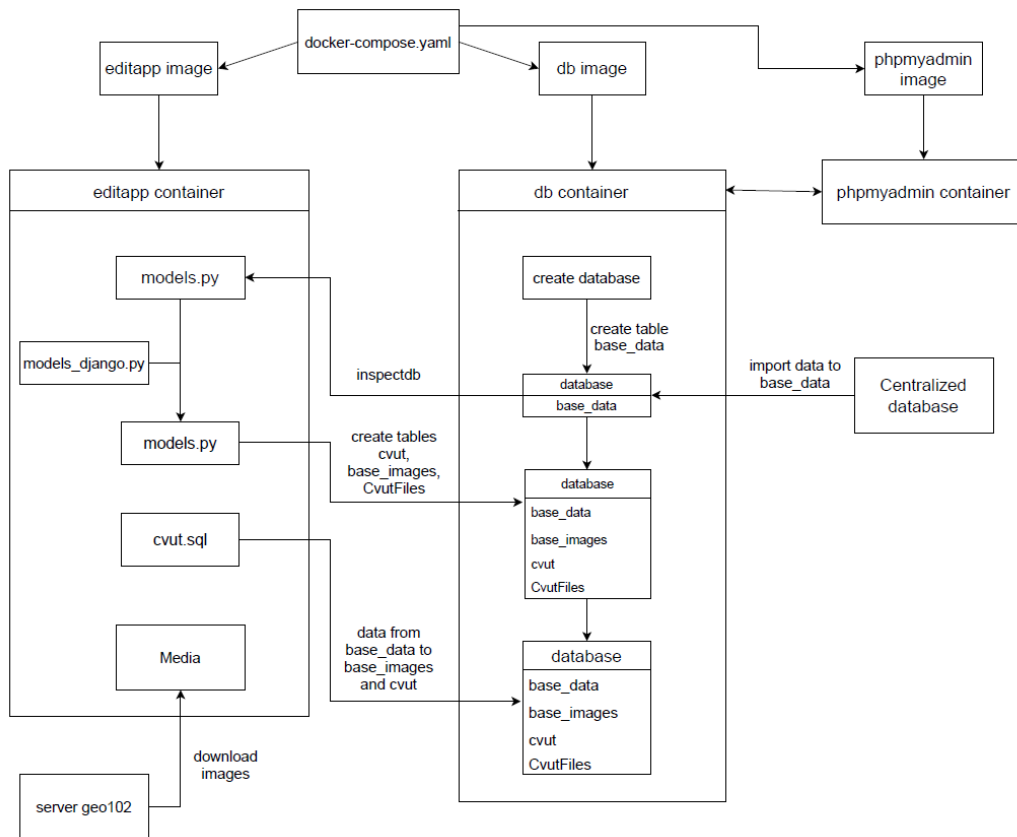
4.12 Automatizace nasazení aplikace a databáze

Úkolem projekčního týmu bylo vytvořit prostředí pro jednotný vývoj aplikace. Do tohoto prostředí bylo ovšem potřeba zahrnout jak aplikaci, tak databázi, ke které se aplikace připojuje. Základním souborem pro tvorbu obrazů je *docker-compose.yml*. Ten definuje základní nastavení pro vytváření tří obrazů, ze kterých se budou spouštět jednotlivé kontejnery. Prvním obrazem je *db*, ze kterého vzniká kontejner, ve kterém běží vytvořená databáze MariaDB. Druhým obrazem je *phpmyadmin*, který se nesestavuje lokálně, ale je stažen z *hub.docker.com*. Třetí obraz je nazván *editapp* a vytváří kontejner, ve kterém běží vyvíjená webová aplikace. U každého obrazu je zde potom definováno, z jakých souborů se obraz vytvoří, název kontejneru, port, na kterém daný kontejner poběží a další nastavení, potřebná pro zajištění správného chodu kontejnerů.

Vytvoření obrazu *db* je definováno z adresáře *mariadb*, kde v souboru *Dockerfile* je přesně popsáno jeho sestavení. Jako první je zde definováno sestavení databáze *mariadb*. Dále je zde doinstalován soubor *requirements.txt*, který obsahuje potřebná rozšíření. Dalším souborem *init.py* se vytváří základní tabulka *Base_Data*, která se striktně vytváří dle aktuální verze centrální databáze a neimituje její změny. S touto tabulkou se pomocí skriptu *sync-db.py* synchronizují data z centrální databáze, a nakonec se pomocí *permissions.sql* vytvoří superuživatel, a nastaví se práva v databázi.

Vytvoření obrazu s webovou aplikací je nastaveno na adresář *editapp*, a je definováno pomocí *Dockerfile*. Definice image je nastavena na Python 3.9, ve kterém také běží Django. Dále se opět musí doinstalovat potřebná rozšíření, zapsaná v *requirements.txt* a spustí se skript *startup.sh*. Tímto skriptem se databáze doplňuje o další tabulky a jejich objekty. Nejprve se zde provede `python manage.py inspectdb`, kterým se vytvoří *models.py* s tabulkou *Base_Data*. Vytvořený model je rozšířen o tabulky *Cvut*, *CvutFiles* a *Base_Images* a migrací je převeden do databáze. Toto je z důvodů autentifikačního systému, kdy *Managers*, které jsou v modelu zapsány v tabulce *Cvut* vytváří v databázi vlastní tabulku. Pokud by se vytvořila tabulka *Managers* v databázi a pomocí `python manage.py inspectdb` by se převedla do modelu, vytvořila by se tam jako samostatná tabulka a nespĺňovala by požadovanou funkcionalitu. Vytvořené tabulky *Cvut* a *Base_Images* se pomocí SQL příkazů v

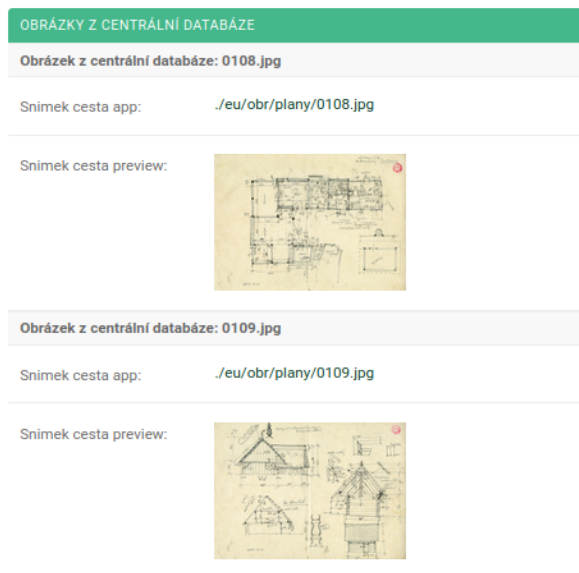
cvut.sql doplní o data z tabulky *Base_Data*. Obrázky na které se odkazují jsou momentálně předpřipravené a uložené na serveru *geo102.fsv.cvut.cz*, odkud se stáhnou a uloží do složky *media*. Nakonec se aplikace spustí příkazem `python manage.py runserver`.



Obrázek 4.14: Postup tvorby aplikace a databáze

4.13 Úprava admin prostředí

S přidáním nové tabulky a nových proměnných bylo potřeba upravit zobrazení v administrátorském prostředí. Data z centrální databáze uživatelé nebudou moci přidávat, editovat ani mazat. U tabulek Cvut a Base_Images byly proto tyto možnosti úplně odstraněny a tabulka Base_Images byla nastavena, aby se její fotografie, které jsou v ní uloženy, zobrazovaly u jednotlivých objektů. U tabulky CvutFiles byla nastavena možnost přidávání, editace nebo mazání souborů administrátorům a uživatelům s oprávněním k danému objektu. Bylo také upraveno zobrazení dat jak v hlavní nabídce, tak na stránce zobrazující informace o daném objektu. U obrázků byl také vytvořen jejich náhled v administrátorském prostředí s možností jejich zvětšení.



Obrázek 4.15: Zobrazení fotografií v administrátorském prostředí

```
class PostBaseAdmin(admin.StackedInline):
    model = BaseImages
    fields = ['snimek_cesta_app', 'snimek_cesta_preview']
    readonly_fields = ['snimek_cesta_app', 'snimek_cesta_preview']

    def snimek_cesta_preview(self, obj):
        if obj.snimek_cesta_app:
            return mark_safe(''.format(obj.snimek_cesta_app.url))
        else:
            return '(No image)'
```

Obrázek 4.16: Vytvoření náhledu v admin.py

Závěr

Cílem práce bylo vytvořit webovou aplikaci pro projekt Viskalia. Aplikace měla být vytvořena tak, aby uměla zobrazovat, vytvářet, editovat a mazat data v databázi. Tento základní předpoklad byl splněn, kdy aplikace splňuje všechny tyto funkcionality i s náležitými úpravami.

Co se týče aplikace, je zde několik možností na zlepšení, popřípadě dodělání nedostatků aplikace. Jedním z nedostatků je pracná tvorba vytváření uživatelů a nastavování jejich práv k objektům, kdy se každému uživateli musí samostatně nastavit práva ke každému objektu, pro který by měl mít právo editace. Toto by šlo vyřešit jednoduchým skriptem pro hromadný zápis práv do tabulky Managers. Dalším nedostatkem je použití balíčku pro editaci administrátorského prostředí *admin_interface*. Aktuálním problémem je chybějící šablona, která by se při sestavování kontejnerů vložila do databáze. Při opětovném vytvoření se tedy vždy restartuje nastavení, které superuživatel vytvoří. Možným řešením je tuto šablonu vytvořit a přidat jí při spouštění do tabulky *admin_interface*, nebo toto rozšíření odstranit a upravit základní HTML šablonu, kterou poskytuje Django. Posledním námětem je dokončení uživatelské aplikace, aby byla uživatelsky přívětivější, ať už se jedná pouze o zobrazování dat, nebo dodělání funkcionalit podobných administrátorskému rozhraní.

Jedním z úkolů projektového týmu Viskalia bylo také automatizovat nasazení aplikace a databáze. Tento problém byl také vyřešen, ovšem obsahuje několik nedostatků, které by bylo potřeba opravit. Prvním nedostatkem je synchronizace struktury databáze a vložených dat, kdy při tvorbě tabulky, která přebírá data z centrální databáze není proveden sken této databáze, ale je přímo vytvořena tabulka, která odpovídá její aktuální verzi. Pokud se tedy provedou změny v této databázi, nepromítnou se do námi vytvořená tabulky. Dalším problémem je stahování obrázků z databázového serveru. Obrázky nejsou stahovány z databázového serveru, ale je stahován pouze předpřipravený soubor s několika vybranými obrázky.

Seznam zkratek

Viskalia	Virtuální skansen lidové architektury
SQL	Standardizovaný strukturovaný dotazovací jazyk (Structured Query Language)
NoSQL	Ne jenom SQL (Not only SQL)
PHP	Hypertextový preprocesor (Hypertext Preprocessor)
LXC	Linuxové Kontejnery (LinuX Containers)
HTML	Hypertextový značkovací jazyk (Hypertext Markup Language)
CSS	Kaskádové styly (Cascading Style Sheets)
ORM	Objektově relační zobrazení (Object-relational mapping)
ID	Hodnota jednoznačně určující každý záznam
URL	Jednotná adresa zdroje (Uniform Resource Locator)
WSGI	Web Server Gateway Interface
ASGI	Asynchronous Server Gateway Interface
HTTP2	Hypertext Transfer Protocol 2
IoT	Internet věcí (Internet of Things)
OS	Operační systém (Operating system)
GUI	Grafické uživatelské rozhraní (Graphic User Interface)
CSV	Hodnoty oddělené čárkami (Comma-separated values)
GIS	Geografický informační systém (Geographic information system)
FGIS	Free software GIS
UTC	Koordinovaný světový čas (Coordinated Universal Time)

Odkazy

- [1] Wikipedia. Flask (web framework). [online], Last updated May 8, 2021. Dostupné z: [https://en.wikipedia.org/wiki/Flask_\(web_framework\)](https://en.wikipedia.org/wiki/Flask_(web_framework)).
- [2] Wikipedia. Django (web framework). [online], Last updated April 13, 2020. Dostupné z: [https://en.wikipedia.org/wiki/Django_\(web_framework\)](https://en.wikipedia.org/wiki/Django_(web_framework)).
- [3] Wikipedia. Mariadb. [online], Last updated May 15, 2021. Dostupné z: <https://cs.wikipedia.org/wiki/MariaDB>.
- [4] Marek Moravcik, Pavel Segec, Martin Kontsek, Jana Uramova, and Jozef Papan. Comparison of lxc and docker technologies. In *2020 18th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, pages 481–486, 2020.
- [5] Melissa Hollingsworth. Data normalization, denormalization, and the forces of darkness. *vol. V. 0.4, ed*, 2017.
- [6] Wikipedia. Object–relational mapping. [online], Last updated May 18, 2021. Dostupné z: https://en.wikipedia.org/wiki/Object-relational_mapping.
- [7] Michael Bayer. Sqlalchemy. In Amy Brown and Greg Wilson, editors, *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*. aosabook.org, 2012.
- [8] Django Software Foundation. Writing your first django app, part 1. [online], 2020. Dostupné z: <https://docs.djangoproject.com/en/3.1/intro/tutorial01/>.
- [9] Django Software Foundation. django-admin and manage.py. [online], 2020. Dostupné z: <https://docs.djangoproject.com/en/3.1/ref/django-admin/>.
- [10] Microsoft Corporation. Django packages. [online], 2020. Dostupné z: <https://djangopackagesorg.readthedocs.io/en/latest/index.html>.
- [11] Django Software Foundation. The django admin site. [online], 2020. Dostupné z: <https://docs.djangoproject.com/en/3.1/ref/django-admin/>.

- [12] A. Mouat. *Using Docker: Developing and Deploying Software with Containers*. O'Reilly Media, 2015.
- [13] Scott Chacon and Ben Straub. *Pro Git*. Apress, USA, 2nd edition, 2014.
- [14] Wikipedia. phpmyadmin. [online], Last updated March 19, 2021. Dostupné z: <https://cs.wikipedia.org/wiki/MariaDB>.
- [15] The phpMyAdmin devel team. phpmyadmin Úvod. [online], 2021. Dostupné z: <https://docs.phpmyadmin.net/cs/latest/intro.html>.
- [16] W.S. Vincent. *Django for Beginners: Build Websites with Python & Django*. Amazon Digital Services LLC - KDP Print US, 2020.
- [17] Shabda Raaj. *Django Admin Cookbook: How to do things with Django Admin*. Agiliq Info Solutions India Pvt Ltd; 1st edition, 2018.

Seznam příloh

A Průvodce aplikací	42
B Zdrojový kód	46

A Průvodce aplikací

Po zadání webové adresy do administrátorského rozhraní se zobrazí přihlašovací okno. Pro úspěšné přihlášení musí být uživatel nastaven jako staff.

Figure A.1: Přihlašování do administrátorské aplikace

Pokud se uživatel přihlásí jako administrátor (superuser), má přístup ke všem funkcím a tabulkám, které Django poskytuje. To je editování vzhledu stránky, přidávání, editování a mazání uživatelů a skupin a přístup do tabulky Cvut s přidruženými tabulkami.

Figure A.2: Hlavní stránka pro přihlášení jako administrátor

Zobrazení stránky pro uživatele, který má přístup pouze k tabulce Cvut.

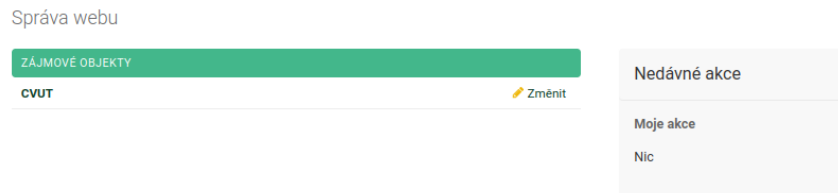


Figure A.3: Zobrazení stránky pro uživatele

Zobrazení tabulky Cvut s filtry a vyhledáváním.

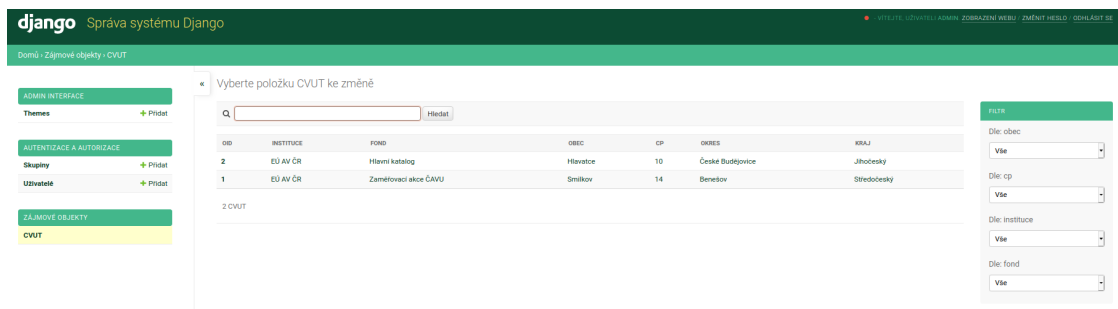


Figure A.4: Hlavní stránka pro přihlášení jako administrátor

Po rozkliknutí záznamu se o něm zobrazí všechna data z tabulek Cvut, Base_Images a CvutFiles. Superuživatelé mají možnost měnit práva k objektům pomocí tabulky Managers.



Figure A.5: Zobrazení záznamu s možností měnit práva uživatelů

Zobrazení fotografií s náhledem z tabulky Base_Images a souborů z tabulky CvutFiles. Možnost přidání a editace souborů je možná, jen pokud k tomu má uživatel oprávnění.

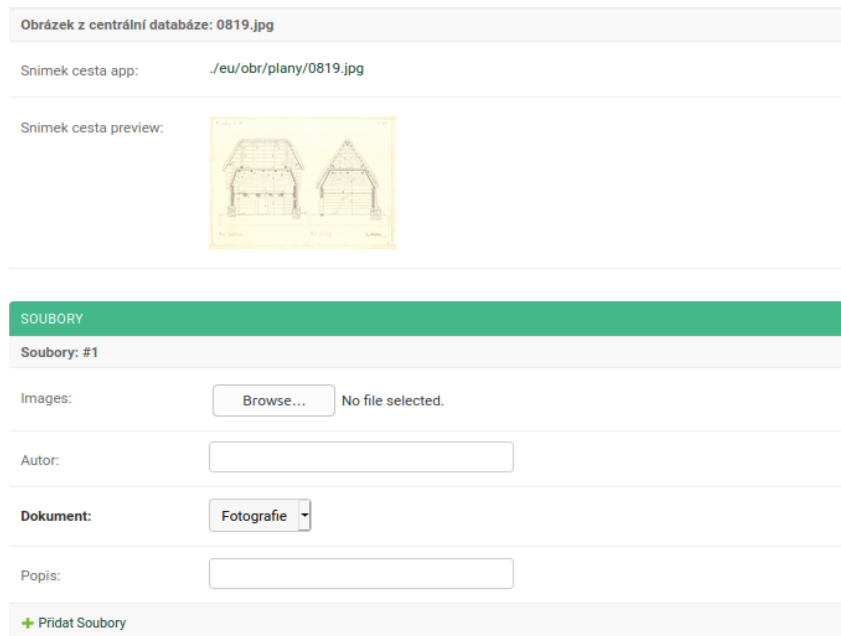


Figure A.6: Zobrazení záznamu s možností měnit práva uživatelů

Po rozkliknutí *Uživatelé* se zobrazí seznam všech uživatelů. Zde je možnost jejich editace, kde lze nastavit administrátorský přístup, superuživatele, zařadit uživatele do skupin nebo mu přiřadit jednotlivá práva.

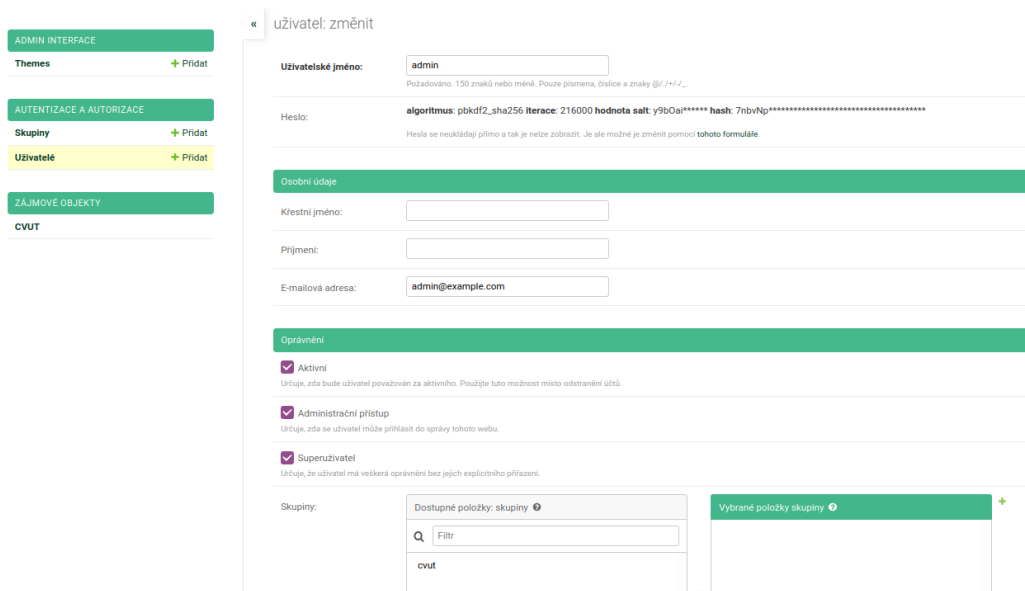


Figure A.7: Správa uživatelů

Možnost nastavení skupin, u kterých lze navolit jednotlivá práva a poté do nich jednoduše přidávat uživatele.

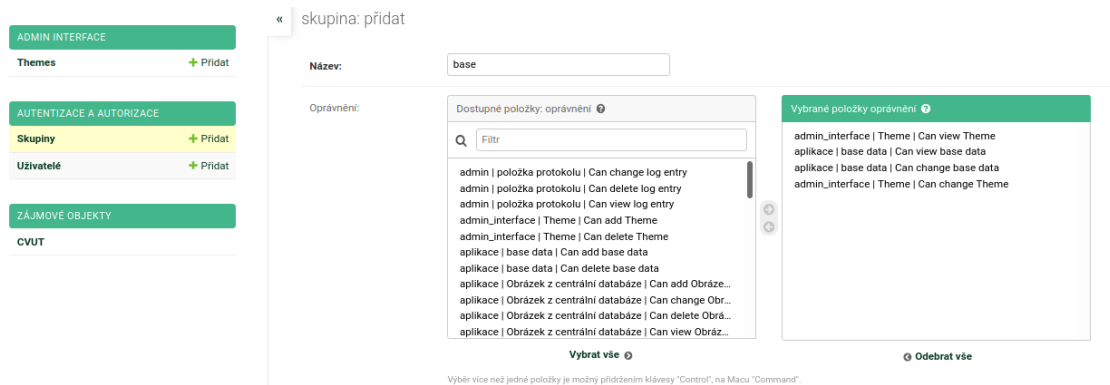


Figure A.8: Tvorba skupin

V nastavení *Theme* se může jednoduše měnit styl zobrazení stránky. Nastavit se dá například logo stránky, barva pozadí nebo textu.

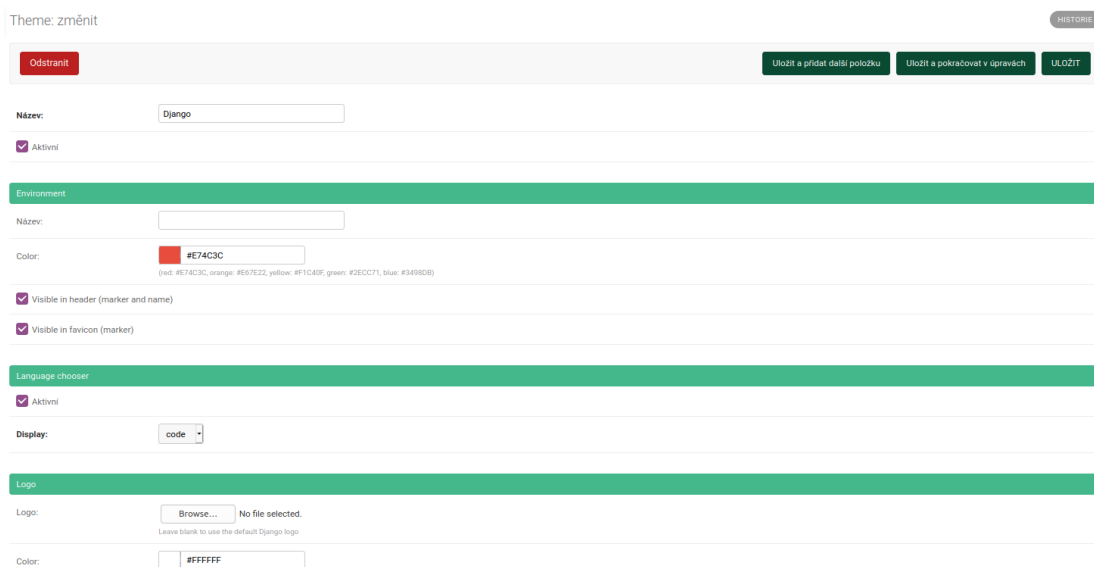


Figure A.9: Nastavení vzhledu stránky

B Zdrojový kód

- naki-viskalia.zip