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

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta stavební
Thákurova 7, 166 29 Praha 6

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: Kulhavý Jméno: Adam Osobní číslo: 440829

Zadávající katedra: Katedra geomatiky

Studijní program: Geodézie a kartografie

Studijní obor: Geomatika

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce: Tvorba webové editorské aplikace pro projekt Viskalia

Název diplomové práce anglicky: Web application for Viskalia project

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: Ing. Martin Landa, PhD.

Datum zadání diplomové práce: 19.2.2021 Termín odevzdání diplomové práce: 17.5.2021
Údaj uveďte v souladu s datem v časovém plánu příslušného ak. roku

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

abstrakt... doplnit

KLÍČOVÁ SLOVA

Python, Django, Webová aplikace.

ABSTRACT

abstract eng.

KEYWORDS

Python, Django, Web application

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 chtěl poděkovat panu Ing. Martinu Landovi, Ph.D. za pomoc při ..... Zvlášť bych chtěl poděkovat celé své rodině .....

Obsah

Seznam obrázků

Seznam tabulek

Úvod

0.1 Úvod

Cílem práce je vytvořit webovou aplikaci pro projekt VISKALIA (Virtuální skansen lidové architektury). 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 veřejností. Aplikace by měla umět zobrazovat data z databáze převzaté z fondů Národního muzea. Ta bude dále rozšířena o námi poskytovaná data jako fotografie, modely, dokumenty a jejich metadata. Tyto data už budou moci uživatelé přidávat, zobrazovat, mazat a editovat. Jeden z hlavních požadavků je také na per object permissions, tedy aby uživatelé mohli editovat pouze taková data, ke kterým mají udělené oprávnění. Aplikace bude po dokončení spuštěna na serveru ČVUT.

1 Rešerše

1.1 Volba metody pro tvorbu webové aplikace

Při volbě, jakou metodu pro tvorbu aplikace použít, se nabízely dvě variant. První variantou by bylo použit skriptovací jazyka PHP a druhou použít python framework pro tvorbu webových aplikací.

Hlavním faktorem při volbě metody zde byla úplná neznalost jazyka PHP, se kterým jsem se během studia ani mimo něj nesetkal. Jasnou volbou tedy bylo použití některého z python frameworků.

1.2 Volba python frameworku

Výběr vhodného frameworku po tvorbu aplikace byl vzhledem k jejich množství opravdu složitý. Mezi hlavní kritéria pro výběr frameworku jsme zařadili snadnou práce s databází, implementovat per object permissions a dostupnost výukových materiálů a komunitní podpora. Nakonec se vybíralo se ze dvou frameworků, a to Flask a Django.

Obrázek 1.1: Flask vs Django []

Flask je open source webový framework napsaný v Pythonu, 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 GitHubu z webových frameworků, a díky tomu disponuje spoustou materiálů a tutoriálů. Avšak velká nevýhoda flasku pro vývoj naší aplikace je, že nedisponuje data 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.

Django je nejpoužívanějším open source frameworkem disponuje objektově relačním mapováním. Jeho skvělou vlastností je tedy možnost generovat model z databáze, který se zde může upravovat a poté jednoduchými příkazy promítnout úpravy do databáze. Další výhodou je implementované administrátorské rozhraní a možnost doinstalováním přídavných balíčků, poskytující jak grafické úpravy, tak přidané funkce. Jedním z nich je i balíček django-guardian zajišťující podporu per object permissions.

Z těchto dvou frameworků jsem nakonec vybral pro tvorbu aplikace Django. To oproti Flasku disponovalo snadnou a rychlou práci s modelem databáze. Další výhodou je vývojáři implementované administrátorské rozhraní, které se nemusí doinstalovat pomocí přídavného balíčku. Z hlediska popularity a dostupnosti výukových materiálu 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, námi rozšířená o další položky. Data zprostředkovaná Národním muzeem byla uložena v MySQL databázi a jednalo se jak o textová, tak obrazová data uložena v tabulkách. 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 použít nějakou z nerelačních databází. Asi zásadní problém s NoSQL databází, který by mohl nastat je problém 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. Zvolili jsme tedy relační databázi a to MariaDB, která je forkem původní MySQL. Od MySQL se liší pouze nepatrně a to přidanými novými funkcemi, odstraněním bugů z MySQL a měla by zajišťovat rychlejší operace s tabulkami.

1.4 Automatizace deploymentu

Jedním z hlavních úkolů je také automatizace deplymentu aplikace a zajištění bezproblémového chodu na všech zařízeních. Při této otázce jsme se nejprve museli rozhodnout, na jakém operačním systému se bude aplikace vyvíjet a následně bude provozována. Po kratší úvaze jsme zvolili systém Linux, který je vhodnější zejména pro vývoj aplikací v jazyce Python. Zde jsme tedy hledali 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 jezdny 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. Daléím rozdílem oproti LXC je, že může být použit i na jiném operačním systému než je linux. Z těchto důvodů byl pro kontejnerizaci zvolen Docker.

2 Použité technologie

2.1 Django

Obrázek 2.1: Django Logo []

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 updatů. V roce 2008 byla vydána verze 1.0. Vývoj se nezastavil a po několika updatech byl v roce 2013 vydán velký update 1.5, který zajišťoval také podporu pro Python 3, dříve pouze Python 2. Dalším velkým updatem 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 velký update v podobě Djanga 4.0, který je plánován na začátek roku 2022.

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 Pytohn 3.6 nebo vyšší. Nejjednodušší je pomocí příkazové řádky zpustit příkaz pip install django, čímž by se měla nainstalovat nejaktuálnější verze Djanga. 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]. Tímto se vytvoří nový adresář který vypadá následovně:

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

__init__.py určuje, aby s adresářem bylo zacházeno jako s packagem.

setting.py zde se definují všechna nastavené Django projektu.

urls.py v projektovém adresáři definuje seznam url adres na úrovní 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 Djanga přidán ve verzi 3.0 a je nástupce staršího WSGI. Výhodou WSGI oproti ASGI je, že dokáže pracovat se složitějšími protokoly jako jsou WsbSocket, 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 spouštění specifických příkazů například pro spuštění aplikace nebo vytvoření aplikací.

Pomocí nástroje manage.py se v Djangu spouští řada důležitých příkazů. Zde je výčet těch nejdůležitějších a jejich popis.

- *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 spuštěném 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 scan modelu a porovná ho 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 pouze 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 *makemigrations*.
- *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 jejích výhodou je, že je lze opětovně použít v dalších projektech. Aplikace se vytvoří příkazem python manage.py startapp [nazev aplikace]. Schéma je následující:

- [Název aplikace]/
  - __init__.py
  - admin.py
  - app.spy
  - 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

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 *makemigrations*

2.1.3 Django – práce s databází

Jednou z nejlepších 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 databází 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 makemigrations a migrate vytvoří nebo upraví tabulky v naší databázi. S databází jako takovou tedy vůbec nemusíme pracovat. Při připojování, k již existující databázi lze příkazem inspectdb vygeneruje model stávající databáze. Django oficiálně podporuje databáze PostgreSQL, MariaDB, MySQL, Oracle a SQLite. Jsou zde ovšem i komunitně vytvořené addony pro podporu NoSQL databáze MongoDB.

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 – packages

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 admin 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*, 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.

2.1.5 Django – admin site

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 udělenu práva k jednotlivým modelům. V aplikaci je možné nastavit práva jednotlivcům, popřípadě skupinám, do kterých se poté uživatelé přidávají.

Registrace modelů a jejich následná customizace se prování 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.

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 uložišti. Jeho užití spočívá v zadokrování aplikace a všech jejich částí. Díky tomu je poté zajištěn bezproblémový chod na jakémkoliv jiném zařízení se stejným operačním systémem. Image pro Linux mohou 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 zadokrování aplikace se nejprve vytvoří image, která obsahuje všechny aplikace a nastavení. Z těchto imagí se poté vytvoří kontejner, který obsahuje také svou vlastní vrstvu, kde jsou uloženy všechny změny, které uživatel provedl.

2.3 Git

Git je open source verzovací systém, který se hojně používá při vývoji aplikací. Jeho použití je možné na všech platformách a jeho 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.

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. Její výhodou je rychlejší zpracování velkého objemu dat nebo podpora různých enginů.

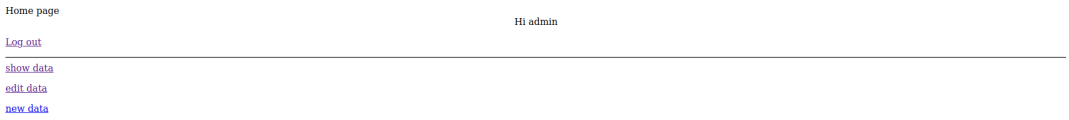
2.5 phpMyAdmin

Je to open source nástroj pro správu MySQL a MariaDB databáze pomocí webového rozhraní. Je napsaný v jazyce PHP a je k dispozici v 80 jazycích včetně češtiny. Toto rozhraní umožňuje provádět SQL příkazy jako jsou CREATE, DROP, ALTER table a spravovat obsah tabulek. Dále je zde možno 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.

3 Seznámení s aplikací

3.1 Navázání na projekt FGIS

Tato diplomová práce navazuje na započatý projekt z předmětu Free Software GIS. Na projektu jsem se účastnil já, Tomáš Lauwerys a Michal Zíma. Projekt měl za úkol vytvořit jednoduchou webovou aplikaci, kdy přihlášení uživatelé mohli zobrazovat, vytvářet, editovat a mazat dat 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 fungovali. Data do databáze šla přidávat, dali se zobrazovat a také mazat, ale při editaci se po uložení nepřepsala aktuální data v databázi.



Obrázek 3.1: Náhled aplikace vytvořené v projektu FGIS

3.2 O aplikaci

V aplikaci jsou použity pohledy založeny na třídách, které se v Pythonu používají pro nahrazení pohledů jako funkcí. Použity jsou základní pohledy jako ListView pro zobrazení většího obsahu tabulky nebo CreateView pro uložení dat do databáze. V aplikaci v soubory *views.py* jsou tedy vytvořeny třídy pohledů pro každou url adresu. V souboru *urls.py* jsou propojeny url adresy, pohledy a html šablony, na které se odkazují. Složka *templates* poté obsahuje html soubory jednotlivých zobrazovaných stránek. Dále je v aplikaci v *models.py* zobrazena struktura celé databáze.

V adresáři projektu je pak v *settings.py* nastaveno připojení k databázi, importována django aplikace, připojen adresář obsahující složku *templates* nebo nastavení domovské adresy. Soubor *urls.py* kromě administrátorské stránky také obsahuje připojení aplikace, kde se tedy odkazuje na soubor *urls.py* v adresáři aplikace.

přiložit obrázky souborů?

4 Tvorba aplikace

4.1 Vytvoření základní aplikace

Aplikace, která byla vytvořena v projektu FGIS nebyla vytvořena správně, a obsahovala zbytečně složitý, a chybný kód. Začalo se tedy od začátku. Nejprve byla 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** se 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, viz. Instalace a inicializace projektu. Nakonec byla provedena migrace, any se v databázi vytvořily Django tabulky.

Na stránce GitLab 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

Prvním větším úkolem bylo vytvořit aplikace, která bude zobrazovat data, a po přihlášení je bude moci uživatel také přidávat, mazat a editovat. Nejprve se tady do adresáře aplikace přidala soubor urls.py, který se následně připojil do projektového adresáře urls.py. Tento způsob je výhodný při použití více aplikace, 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.

```python
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 se provede 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.

```python
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.

```
{% 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 by se neměly ukládat do databáze, ale do adresáře v aplikaci. Databáze poté bude 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 byl 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.

```
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 obrázk

V nastavení se dále určí rootovský 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 v *settings* se 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.

4.5 Nastavení jazyka a času

Dalším nastavením bylo použití našeho středoevropského času a českého jazyka. Velice jednoduché nastavení, kdy v *settings* 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'**.

Seznam příloh