Objektno relacijsko mapiranje u Django web okolini

Horvat, Lorena

SVEUČILIŠTE U ZAGREBU
FILOZOFSKI FAKULTET
ODSJEK ZA INFORMACIJSKE I KOMUNIKACIJSKE ZNANOSTI
Ak. god. 2020./2021.

Lorena Horvat

Objektno relacijsko mapiranje u Django web okolini

Završni rad

Mentor: dr. sc. Vedran Juričić, doc.

Zagreb, srpanj 2021.

Izjava o akademskoj čestitosti

Izjavljujem da je ovaj rad rezultat mog vlastitog rada koji se temelji na istraživanjima te objavljenoj i citiranoj literaturi. Izjavljujem da nijedan dio rada nije napisan na nedozvoljen način, odnosno da je prepisan iz necitiranog rada, te da nijedan dio rada ne krši bilo čija autorska ili druga srodna prava. Također izjavljujem da nijedan dio rada nije korišten za bilo koji drugi rad u bilo kojoj drugoj visokoškolskoj, znanstvenoj ili obrazovnoj ustanovi.

1. Uvod

Baza podataka je centralni dio svakog informacijskog sustava. Podacima se radi učinkovitosti dohvaćanja i manipulacije ne pristupa direktno, nego se koristi sustav za upravljanje bazom podataka. Navedeni sustav je uglavnom realiziran kao posebna komponenta i nije integriran u sam proizvod, bez obzira radi li se o Windows, Web ili Cloud aplikaciji. Budući da su podaci fizički odvojeni od same aplikacije, javlja se problem pouzdanog i djelotvornog načina razvoja aplikacije. S jedne strane, razvojni inženjeri moraju imati što veću mogućnost kontrole nad operacijama i podacima, a s druge strane ne smiju trošiti previše vremena na razvoj navedenih komponenti.

U radu će se definirati objektno relacijsko mapiranje (engl. *Object Relational Mapping, ORM*), kao jedan od načina na koji se može riješiti navedeni problem. U sklopu rada razvijen je sustav, odnosno aplikacija za provođenje ankete, u Django razvojnom okviru i programskom jeziku Python. Proces izrade aplikacije prikazan je postepeno, a u samom su procesu vidljivi konkretni primjeri implementacije ORM-a.

U drugom poglavlju pod naslovom „Baze podataka“ ukratko su definirani i nabrojani modeli podataka te je napravljen kratak pregled karakteristika i mogućnosti relacijskih i nerelacijskih baza podataka. Treće poglavlje, „Tehnologije“, opisuje alate, koncepte i komponente kojima je moguć razvoj i posluživanje web aplikacija pisanih u programskom jeziku Python. U četvrtom poglavlju pod naslovom „Pristup bazi podataka i podacima“ iznesena je problematika pristupa bazi podataka iz aplikacije i prikazana su neka od mogućih rješenja. Peto poglavlje pod naslovom „Django ORM“ detaljno opisuje navedeni ORM, njegove mogućnosti i sučelja, uključujući modele, migracije i serializatore. U šestom poglavlju, „Web aplikacija“, opisane su funkcionalnosti i mogućnosti web aplikacije izrađene u sklopu rada, koja demonstrira pristup bazi podataka pomoću ORM-a.

2. Baze podataka

Baza podataka je skup međusobno povezanih podataka organiziranih prema logičkim, unaprijed određenim pravilima. Podaci moraju istovremeno biti dostupni većem broju korisnika, uključujući administratorske alate ili korisničke aplikacije, zbog čega se bazi podataka ne pristupa direktno, već putem zajedničkog softvera, odnosno sustava za upravljanje bazom podataka, SUBP (engl. *Database Management System – DBMS*) (Manger, 2003).

Osim istovremenog i efikasnog pristupa podacima, sustavi za upravljanje bazama podataka brinu se za očuvanje integriteta, odnosno za očuvanje pravila i ograničenja nad podacima. Također, korisnici koji pristupaju podacima imaju različite role i ovlasti, a SUBP uključuje sigurnosne mehanizme kontrole i zaštite pristupa (Ramakrishnan & Gehrke, 2011).

Logička organizacija, odnosno skup pravila koji definira strukturu baze podataka naziva se model podataka. Jedan od modela podataka je hijerarhijski model, koji podrazumijeva isključivo odnose u kojem je jedan zapis nadređen, odnosno podređen drugom zapisu, a baza podataka je predstavljena stablom. Mrežni model je sličan hijerarhijskom, ali dozvoljava postojanje više nadređenih zapisa, a baza podataka je predstavljena usmjerenim grafom. Objektni model podrazumijeva postojanje objekata, koji imaju svojstva i metode, a potaknut je prednostima i mehanizmima objektno-orijentiranih jezika. Hijerarhijski i mrežni modeli intenzivno su se koristili 70-ih godina, a objektni model nije prihvaćen koliko se očekivalo (Berg, Seymour & Goel, 2012). Danas se najčešće koristi relacijski model podataka, koji je opisan u sljedećem poglavlju.

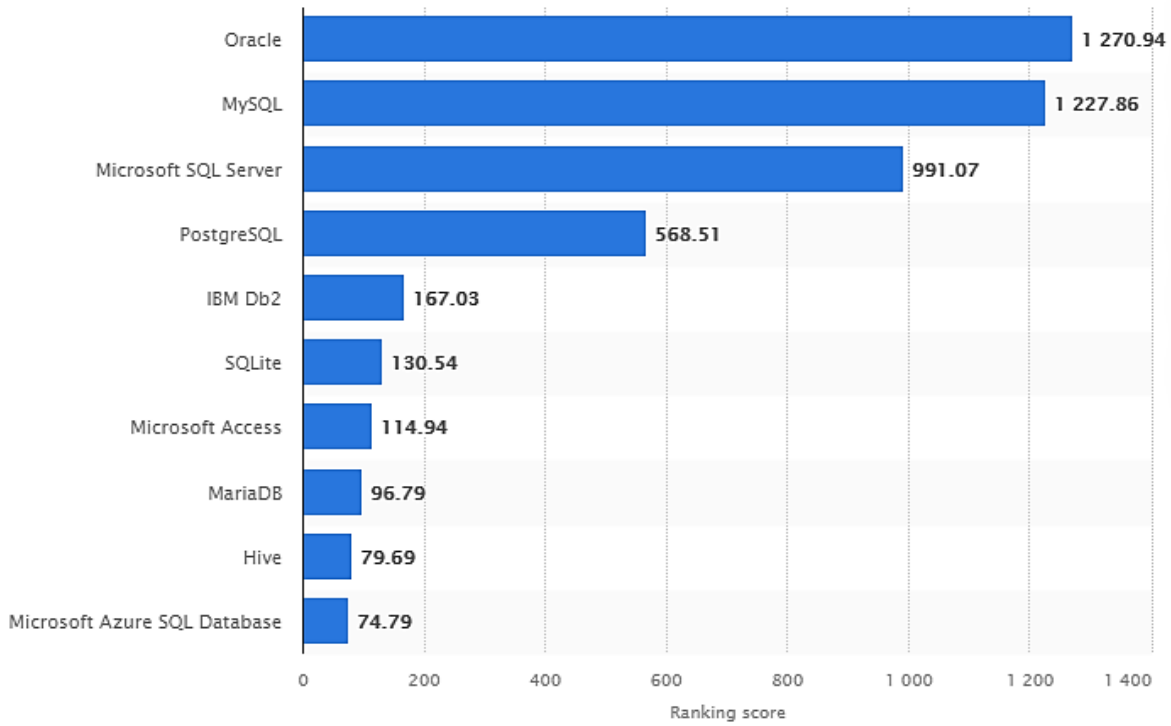
2.1. Relacijske baze podataka

Teoriju relacijskog modela podataka razradio je Codd 1970. u svom radu „A Relational Model of Data for Large Shared Data Banks“ (Codd, 2002). Temeljni elementi modela su atributi, relacijske sheme, n-torke i relacije. Atribut je opis jednog svojstva entiteta, npr. Naziv artikla. Relacijska shema je skup atributa s ograničenjima vrijednosti, odnosno domenom. Na primjeru artikla, relacijsku shemu Artikl mogu činiti atributi (Identifikator, Naziv, Cijena). N-torka je preslikavanje iz svakog pojedinog atributa relacijske sheme u njihove dozvoljene vrijednosti. Primjer n-torke za relacijsku shemu Artikl je (2, „Krug“, 9.50 kn). Relacija je skup n-torki, što znači da u relaciji ne mogu postojati dvije jednake n-torke, odnosno dvije n-torke s jednakim vrijednostima svih atributa.

Na svakoj relacijskoj shemi mora biti definiran barem jedan ključ. Ključ je atribut ili skup atributa koji jednoznačno određuje n-torku u relaciji. Drugim riječima, dvije n-torke ne mogu imati jednake vrijednosti svih atributa ključa. Npr. za relacijsku shemu Osoba (OIB, Ime, Prezime, DatumRodjenja, JMBG) ključevi su OIB i JMBG jer ne postoje dvije osobe s jednakom vrijednosti navedenih atributa. Jedan od ključeva se proglašava primarnim ključem, dok su ostali ključevi alternativni. Primarni ključ je specifičan po tome što sudjeluje u vezama navedene relacije s drugim relacijama, na kojima se povezuje sa stranim ključevima. Veze mogu biti 1:1 (*one to one*), 1:N (*one to many*) i N:N (*many to many*). 1:1 je iznimno rijetka veza koja povezuje entitet u jednoj relaciji s jednim entitetom u drugoj relaciji. 1:N povezuje entitet u jednoj relaciji s više entiteta u drugoj relaciji, npr. odnos između mjesta i osoba. N:N povezuje više entitet u jednoj relaciji s više entiteta u drugoj relaciji, npr. odnos između predmeta i izvođača.

U implementaciji relacijskog modela podataka, odnosno relacijskoj bazi podataka atributu odgovara kolona, relacijskoj shemi skup kolona, n-torci odgovara redak, a relaciji u širem smislu tablica. Zbog toga se relacijska baza podataka sastoji od skupa tablica, koje mogu, ali ne moraju biti povezane. Vrlo je rasprostranjena i prihvaćena zbog toga što relacijski model može opisati većinu problemskog područja, ali i zbog toga što moderni sustavi za upravljanje bazama podataka osiguravaju brzinu rada s podacima, ali i podržavaju SQL, koji omogućuje jednostavno filtriranje i spajanje podataka iz više tablice, korištenje funkcija nad podacima, grupiranje itd.

Od 1970. do danas razvijene su brojne relacijske baze podataka, a 10 najpopularnijih, prema podacima za srpanj 2021., prikazano je na slici 1. Oracle je najpopularnija baza podataka, usprkos činjenici da je besplatna verzija iznimno limitirana u mogućnostima, a iznos komercijalne verzije se kreće od 17.500$ do 47.000 (Oracle.com, 2021). Podržava i određene nerelacijske modele podataka, a u zadnjoj verziji fokusira se na računarstvo u oblaku. MySQL je inicijalno bila baza podataka otvorenog koda, ali trenutno je u vlasništvu Oraclea. Dostupna je besplatna verzija koja obuhvaća najčešće korištenu funkcionalnost, tako da za manje sustave nema potrebe za licencom, čija se cijena kreće od 2.000$ do 10.000$ (Mysql.com, 2021). SQL server je baza podataka u vlasništvu Microsofta. Slično kao i kod MySQL-a, dostupna je vrlo dobra besplatna verzija s ograničenjima poput najveće veličine baze od 10 GB, koja najčešće nisu važna za manje sustave. Microsoft ima vrlo složene cijene licenci, ali cijena Enterprise licence je otprilike 13000$, a Standard licence 3500$ po jezgri procesora (Microsoft, 2021).



Slika 1 - Najpopularnije relacijske baze podataka (Liu, 2021)

PostgreSQL baza je, prema popularnosti, na četvrtom mjestu. Otvorenog je koda, nalazi se pod vrlo slobodnom licencom sličnom MIT licenci i ne postoji komercijalna verzija. Razvija se više od 30 godina i trenutna, 13. verzija, izdana je u rujnu 2020., a u trenutku pisanja ovog rada u najavi je PostreSQL 14 beta 2 verzija (Postgresql, 2021). Kao što je vidljivo iz slike 1., ostale baze podataka, poput IBM Db2, SQLite i Microsoft Access, imaju zamjetno manju popularnost u odnosu na prve četiri.

Najpopularnije baze podataka se, naravno, razlikuju u značajkama i funkcionalnosti, međutim imaju zajednički skup značajki bez kojih se ne bi mogle koristiti u trenutnim sustavima i u trenutne svrhe. Omogućuju rad s raznim tipovima podataka za rad s brojevima, tekstom, geografskim podacima i JSON-om, a omogućuju i kreiranje vlastitog tipa podatka. Omogućuju kreiranje indeksa i ograničenja nad podacima, mjerenje različite statistike, rad s pogledima, rad sa sigurnosnim kopijama, replikaciju, rad sa rolama, korisnicima i dozvolama, kreiranje procedura i funkcija, internacionalizaciju itd.

Međutim, jedno od najvažnijih karakteristika koje bi trebale podržavati moderne relacijske baza podataka su ACID svojstva (engl. *Atomicity Consistency Isolation Durability properties*) (Yu, 2009). Atomarnost (engl. *Atomicity*) se odnosi na sposobnost baze podataka da izvede ili sve operacije ili nijednu operaciju unutar transakcije. Drugim riječima, ako se transakcija sastoji od

10 naredbi, tada baza podataka mora ili uspješno izvesti sve operacije ili se mora vratiti u stanje kakvo je bilo prije izvođenja transakcije. Konzistentnost (engl. *Consistency)* se odnosi na pravila i svojstva baze podataka, koja moraju biti sačuvana bez obzira na to je li transakcija provedena uspješno ili ne, odnosno stanje u bazi mora biti konzistentno i prije i nakon transakcije. Ako je nad tablicom kreiran ključ, tada se baza nakon transakcije ne može dovesti u stanje koje ne bi bilo u skladu s navedenim pravilom. Izolacija (engl. *Isolation*) je svojstvo koje nalaže da dvije operacije ili transakcije moraju biti međusobno nezavisne, što se najčešće postiže zaključavanjem promijenjenih resursa. Ako dvije istovremene transakcije mijenjaju isti redak u tablici, tada to može napraviti samo jedna transakcija, a druga mora čekati sve dok se redak ne otključa. Izdržljivost (engl. *Durability*) se odnosi na trajno očuvanje promjena nastalih nakon uspješno završene transakcije. Ako se transakcijom kreira novi redak u tablici, tada on ostaje u bazi podataka i u slučaju neposrednog pada sustava.

2.2. Nerelacijske baze podataka

Nerelacijske baze podataka omogućuju pohranu, distribuciju i pristup podacima metodama različitim od relacijske baze. Javile su se zbog ograničenja relacijskih baza podataka prilikom rada s velikom količinom podataka i u okruženju u kojem bazu mora koristiti velik broj korisnika. Velike kompanije poput Googlea, Amazona, Facebooka i LinkedIna su prve prepoznale navedene nedostatke i razvile nerelacijske ili NoSQL baze podataka (Gyorödi, Gyorödi & Sotoc, 2015). Za razliku od relacijskih, nerelacijske baze imaju implicitnu shemu podataka, što znači da shema nije zadana ili definirana unaprijed, već se o shemi može zaključiti na temelju spremljene strukture podataka (Ardeleanu, 2016).

Nerelacijske baze podataka omogućuju istovremeno čitanje i promjenu podataka uz vrlo malu latenciju, i vrlo efikasan način spremanja i dohvaćanja velikih količina podataka. Uz to, vrlo su skalabilne, odnosno lako ih je proširiti u skladu s potrebama za resursima ili brojem korisnika [Han et al, 2011).

Baze podataka, odnosno sustavi za upravljanje bazama podataka se klasificiraju u četiri osnovne kategorije: ključ-vrijednost, stupčani, dokument i graf sustavi (Moniruzzaman & Hossain, 2013). Ključ-vrijednost (engl. k*ey-value database*) su najjednostavnije baze podataka, u kojima određenom ključu pripada jedna vrijednost. Zbog svojih iznimno dobri performansi koriste se i u velikim sustavima poput Amazona, koji koristi Dynamo bazu podataka, i LinkedIna, koji koristi Voldemort. Stupčane baze podataka (engl. c*olumn database*) ne spremaju podatke za određeni entitet u obliku retka, nego u obliku niza kolona ili familije

kolona s istim identifikatorom. Jedna od najpopularnijih je Cassandra, besplatna baza podataka otvorenog koda, koju koriste Facebook, Instagram i Netflix. Dokument baze (engl. d*ocument database*) su dizajnirane za upravljanje i spremanje datoteka, poput tekstualnih dokumenata, a jedna od najpopularnijih je MongoDb. Graf baze (engl. *graph database*) omogućuju rad s podacima o entitetima i njihovim međusobnim odnosima, a koriste se kad je odnos među entitetima izuzetno važan. Koriste se npr. u Amazonu, Facebooku i Yahoou, a jedna od najpopularnijih je Neo4j.

Relacijska baza podataka koja se detaljnije opisuje u radu i koristi u samoj aplikaciji je PostgreSQL, najčešće korištena relacijska baza u Django web okolini.

3. Tehnologije

U ovom poglavlju su opisane tehnologije i razvojni okviri koji se koriste za izradu web aplikacije. Budući da je pisana u Pythonu, objektno-orijentiranom jeziku, ukratko su opisane najvažnije karakteristike navedene vrste programskih jezika. Kao razvojni okvir korišten je Django, temeljen na MVC obrascu arhitekture (engl. *design pattern*), a aplikacija se poslužuje pomoću NGINX i PythonAnywhere sofvera.

3.1. Python

Python je objektno orijentiran programski jezik opće namjene te jedan od najpopularnijih programskih jezika današnjice. Prednosti su mu vrlo čitljiva i jasna sintaksa s fokusom na urednost, bogata standardna biblioteka, velik broj biblioteka otvorenog koda, vrlo široko područje primjene i jednostavne skripte (Python.org. 2021).

Objektno orijentirano programiranje predstavlja jedan od temeljnih koncepata razvoja aplikacije i pisanja koda, koji se ne oslanja isključivo na funkcije. Objektno orijentiran programski jezik pretpostavlja postojanje objekata koji se kreiraju pomoću klasa. Dakle, umjesto nizanja funkcija koje obavljaju određenu zadaću, ovaj pristup zahtijeva dijeljenje funkcionalnosti u klase. Klase sadrže podatke i metode, a kada se želi koristiti funkcionalnost klase, na temelju nje je prethodno potrebno kreirati objekt. Zbog navedenog, kod pisan na ovakav način je kompleksniji i glavna namjena je uglavnom razvoj složenijih softverskih sustava.

Objektno orijentirani jezici temelje se na tri osnovna mehanizma: nasljeđivanje, polimorfizam i enkapsulacija. Nasljeđivanje definira strukturu i hijerarhijski odnos između tipova podataka, odnosno klasa. Klasa može naslijediti drugu klasu (tzv. JE odnos), čime nasljeđuje njezina svojstva, metode i općenito, funkcionalnost. Navedena klasa tada proširuje funkcionalnost nadklase, s novim metodama i svojstvima, ili joj može promijeniti funkcionalnost. Polimorfizam je usko vezan uz nasljeđivanje, a odnosi se na mogućnost da se jedan objekt ponaša na više načina, ovisno o samom kontekstu. Enkapsulacija je mehanizam koji sakriva nepotrebne detalje neke klase ili metode. Npr. ako klasa sadrži metodu za slanje maila, tada korisnik navedene klase ne treba znati detalje protokola, mrežne komunikacije, pogrešaka i sl., već je dovoljno da metodu pozove s potrebnim parametrima i da metoda eventualno vrati određeni tip podatka.

DRY (engl. *don't repeat yourself*) označava pravilo kvalitetnog pisanja koda koje pomaže organizaciji koda u manje logičke strukture, olakšava kontrolu nad kodom i pomaže da elementi koda ne utječu jedni na druge. Iako primjenjivo na svaki programski jezik, u kontekstu čitljivosti Pythona i OOP prirode Djanga, ovaj princip je posebno naglašen, kao jedna od glavnih smjernica pisanja koda. Slijedeći DRY princip, OOP strukturira podatke u klase koji se na apstraktan način mogu dodatno podijeliti u logičke komponente. Na taj način svaka komponenta ima jedinstvenu ulogu koja omogućava lakše čitanje, razumijevanje te pisanje samog koda. Nadalje, smanjuje kompleksnost cijele aplikacije te se vrlo dobro uklapa u razvojni model Djanga.

Budući da se aplikacija sastoji od nezavisnih logičkih komponenti, one se mogu preuzeti od drugih korisnika na projektu ili šire zajednice. Glavni Python repozitorij takvih komponenti, odnosno paketa je PyPI. Popularnost i široku primjenu Python programskog jezika dodatno podupire veličina ovog repozitorija jer trenutno sadrži preko 300,000 paketa i ima oko 500,000 aktivnih korisnika. Na PyPI je moguće pronaći paket za gotovo svaku primjenu i kao takav se izvrsno uklapa u Djangov mentalitet brzine i jednostavnosti (The Python Package Index, 2021).

3.2. Razvojni okviri

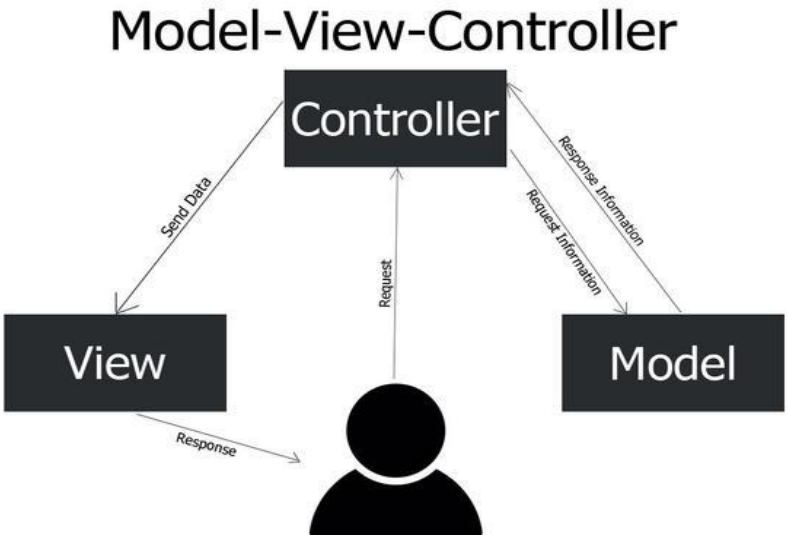
Web razvojni okviri (engl. *framework*) definiraju se kao skup softverskih paketa čija je svrha pružanje niza funkcionalnosti za izradu web aplikacija, a sadrže baze podataka sa svim njihovim funkcijama te grafičko korisničko sučelje.

Jedan od najpopularnijih ORM sustava na svijetu je Django. Često nazivan razvojnim okvirom za perfekcioniste s rokovima, ovaj softver koristi se kao temelj za mnoge popularne aplikacije, poput: Instagrama, Mozille, Pinteresta, Spotifya i National Geographica. S velikim rasponom dodatnih paketa i korisničke podrške, Django dolazi sa svim funkcionalnostima koje su potrebne za bilo kakav oblik razvoja web aplikacija. Nadalje, Django stavlja fokus na sigurnost i zaštitu podataka te je zbog svoje jednostavnosti vrlo lako razvijati i zahtjevnije sustave s kompleksnijim kodom.

Također popularan ORM pisan Python kodom je Flask. Sličan Djangu, ali mnogo jednostavniji za razvijanje i implementaciju. Preporuča se za izradu jednostavnijih aplikacija ili za korisnike koji nisu upoznati s Django razvojnim okvirom i ne preferiraju rad unutar pravila njegovog sustava.

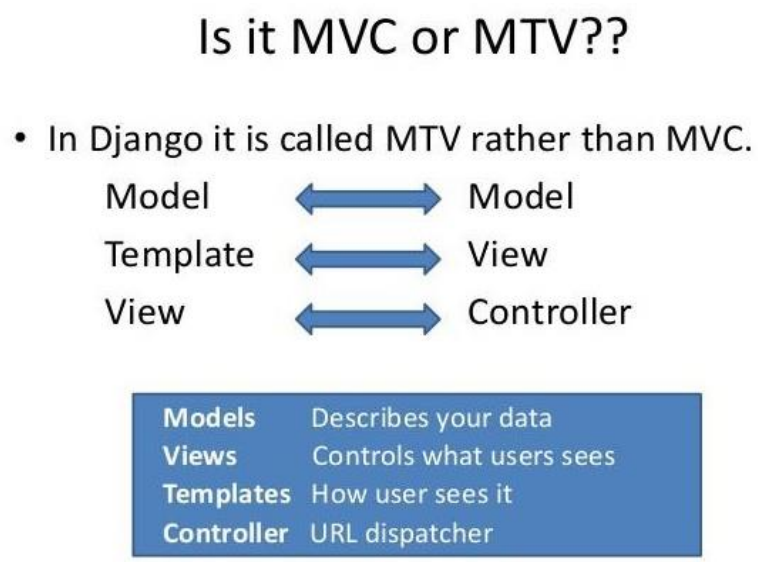
Kao alternativu Pythonu, može se navesti Ruby On Rails, glavnu platforma za razvoj aplikacija pisanih Ruby jezikom. Konceptualno je sličan Djangu, no Ruby On Rails je prilično limitiran zbog nedostatka razvojnih mogućnosti. S mnogo strožim pravilima strukture koda, Ruby On Rails žrtvuje dodatne funkcionalnosti u svrhu bržeg pisanja koda (Singh, 2021).

MVC (engl. *Model-View-Controller*) je arhitektura web aplikacije sastavljena od triju komponenti: model, pogled (engl. *view*) i kontroler (engl. *controller*). Njihova međusobna ovisnost prikazana je na slici 3. Model (*models.py*) je središnji dio aplikacije koji sadrži svu potrebnu logiku i nije ovisan o ostalim komponentama. Jedna od njegovih funkcija je rad s objektima iz baze podataka. View (*views.py* + *templates*) predstavlja korisničko sučelje. To je vizualni, prezentacijski dio aplikacije, koji uključuje liste, tablice, forme, stilove i sl., i između ostaloga služi korisničkom unosu podataka. Controller (*urls.py*) je veza između modela i pogleda. Jedna od njegovih funkcija je validacija i interpretacija podataka s pogleda, priprema modela, izvršavanje određenih operacija nad modelom i vraćanje novog pogleda ili nekog drugog rezultata korisniku.



Slika 2 - MVC arhitektura (Bouhsine, 2021)

Cjelokupni MVC koncept predstavlja način na koji kontroler i view te model i view komuniciraju. Model čine statični podaci u bazi, view predstavlja Django funkcionalnu logiku koju primjenjujemo na aplikaciju, a kontroler služi povezivanju tih dviju komponenata.



Slika 3 - Django MVC implementacija (Oyom, 2017)

Django ne koristi standardnu MVC već MTV arhitekturu, kao što je prikazano na slici 3. Django kao razvojni okvir pretpostavlja da svi zahtjevi prema serveru očekuju HTML/CSS format. Zato je svaki odgovor servera HTML response. U ovom slučaju Django templates predstavljaju prezentacijsku logiku koju im omogućava Jinja jezik; uz dinamično omogućavanje strukturiranja HTML datoteka, poziva i jednostavnije funkcije za dinamičan prikaz informacija. Važno je napomenuti da REST API komponenta Djanga; Django REST framework pruža standardni MVC flow. Pomoću middleware modula koji provjerava svaki zahtjev prema serveru, moguće je replicirati kontroler aspekt MVC arhitekture.

Jedna od glavnih značajki Djanga uključuje zaštitu protiv većine modernih napada s kojom dolazi. Navedeno omogućava potpuno posvećivanje glavnoj problematici aplikacije od strane programera bez brige oko sigurnosnih rizika. Glavne metode napada koje Django sprječava su:

Cross site scripting ili *XSS* - vrsta napada koju karakterizira spremanje nefiltriranih podataka u bazu ili se unose u formu. XSS napad proslijeđuje validan Javascript kod koji, nakon što se dohvati za prikaz, izvršava svoje upute. Django filtrira sve unose koji počinju sa < > znakovima, kao i generalnu regex provjeru generalnih komandi.

Cross site request forgery - napad koji će POST-ati formu na server s podacima koji ne dolaze iz istog izvora. Django svakoj formi dodjeljuje token koji se prilikom unosa te forme provjerava na serveru.

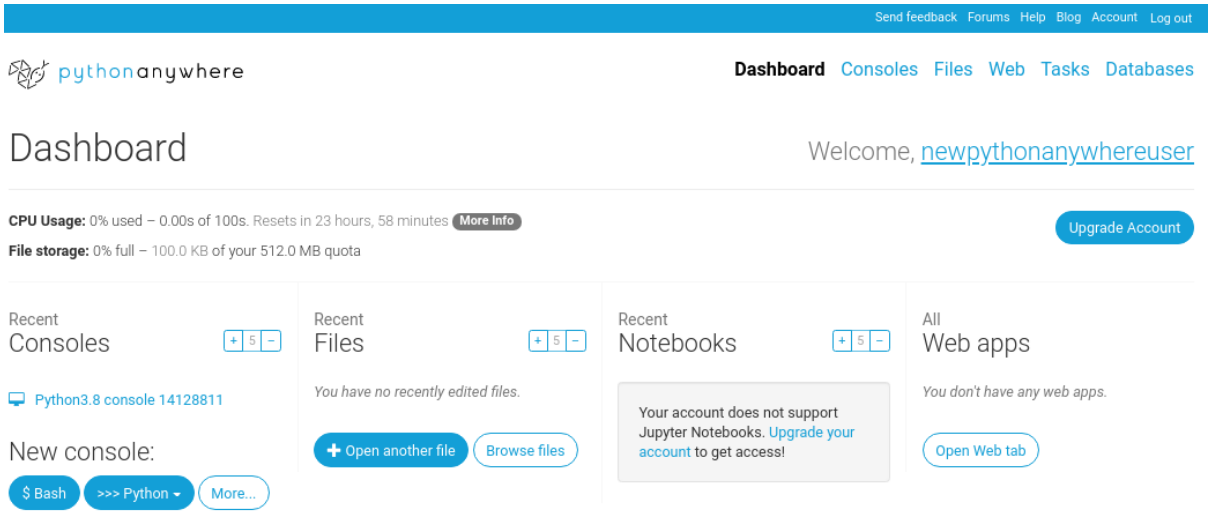
SQL injection - slična vrsta napada XSS-u. Metoda je ista; ubacivanje SQL koda umjesto korisničkih podataka i pokušaj izvršavanja brisanja ili ubacivanja podataka u bazi servera.

Django štiti i od ostalih generičnih napada poput *clickjacking protection*, *SSL/HTTPS*, *host header validation*, *referer policy* i *session security*.

3.3. Posluživanje aplikacija

NGINX predstavlja *open source* tehnologiju čija primarna svrha uključuje posluživanje aplikacija na nižoj, sustavnoj razini. Često hvaljen u pogledu dugotrajnosti i neometanom odnosu s aplikacijom**,** također predstavlja najjednostavniji izbor kod serviranja Django aplikacija. U komunikaciji sa *uwsgi.py* modulom, NGINX osigurava dostupnost aplikacije na specifičnoj domeni, preusmjeravanje zahtjeva na specifične poddomene, konfiguraciju mail klijenata te raspoređivanje prometa na više instanci servera putem *load balancera* (NGINX, 2021).

PythonAnywhere cloud platforma predstavlja najjednostavniji odabir prilikom serviranja Django aplikacije. Automatski postavlja glavni direktorij, domenu, poveznicu sa NGINX-om i statičnim datotekama. Dodatno, znatno olakšava podizanje određene aplikacije (PythonAnywhere, 2021). Na slici 4 prikazan je primjer PythonAnywhere platforme.



Slika 4 - PythonAnywhere cloud platforma

Jednostavno korisničko sučelje osigurava jednostavnije dodavanje resursa serveru, mijenjanje koda, promjenu domene ili dodavanje nove aplikacije i poddomene. Osim kroz korisničko

sučelje, moguće je koristiti i programsko sučelje, odnosno naredbeni redak, koji osigurava pokretanje Django komande direktno na serveru. Unaprijed definirane akcije moguće je implementirati putem Tasks forme. Budući da je sučelje dostupno preko preglednika, znatno olakšava pristup i kontrolu servera. Korisnik više nije ovisan o specifičnom računalu ili operacijskom sustavu. Ovakav *Platform as a Service* pristup je brz i jednostavan način za posluživanje manjih Python aplikacija.

4. Pristup bazi podataka i podacima

Četiri osnovne metode manipulacije podacima u bazi podataka, odnosno njihove trajne pohrane su stvaranje, čitanje, ažuriranje i brisanje podataka (engl. C*reate, Read, Update, Delete – CRUD*). U modernim sustavima je baza podataka fizički odvojena od ostalih komponenti, što znači da ne mora nužno biti na istom računalu na kojem je npr. web aplikacija. Jedini način na koji aplikacija može komunicirati, odnosno dohvaćati i mijenjati podatke, ali i strukturu baze, je korištenje SQL-a.

SQL (engl. *Structured Query Language*) je upitni skriptni jezik korišten u relacijskim bazama podataka. 1986. godine postao je standardnim programskim jezikom Američkog nacionalnog instituta za standarde. Osnovni skup naredbi i sintaksu podržavaju gotovo sve relacijske baze podataka. Naredba za umetanje podataka u tablicu koja je napisana za jednu bazu podataka će se uz minimalne promjene moći pokrenuti nad bilo kojom drugom bazom.

Uz instalaciju baze podataka uglavnom dolazi i administratorski alat koji omogućuje direktan rad s bazom podataka kroz naredbeni redak (engl. *command prompt*). Slika 5 prikazuje kako se korištenjem *interactive shell* alata u PostgreSQL bazi, uz pomoć SQL-a ručno se može stvoriti nova baza podataka.

```
postgres=#
postgres=# CREATE USER mysite WITH PASSWORD 'mysite';
CREATE ROLE
postgres=# CREATE DATABASE mysite WITH OWNER mysite;
CREATE DATABASE
postgres=#
```

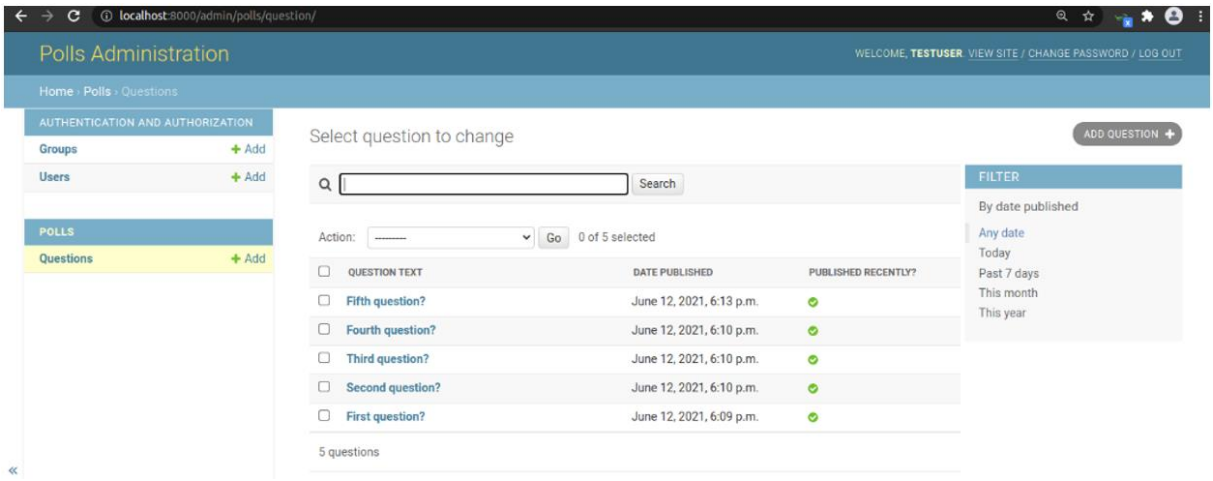
Slika 5 - SQL CREATE naredba

Na isti način moguće je i dohvaćanje podataka. Slika 6 prikazuje primjer pokretanja SQL SELECT naredbe nad jednom tablicom, zajedno s rezultatima u obliku tablice.

```
mysite=# SELECT "polls_question"."id", "polls_question"."question_text",
"polls_question"."pub_date" FROM "polls_question";
 id |  question_text   |          pub_date
----+------------------+------------------------
  1 | First question?  | 2021-06-12 18:09:42+02
  2 | Second question? | 2021-06-12 18:10:09+02
  3 | Third question?  | 2021-06-12 18:10:32+02
  4 | Fourth question? | 2021-06-12 18:10:51+02
  5 | Fifth question?  | 2021-06-12 18:13:57+02
(5 rows)
```

Slika 6 - SQL SELECT naredba

Osim naredbenog retka, baze podataka uglavnom imaju i administratorski alat s grafičkim korisničkim sučeljem, koji omogućuje osnovni rad s bazom i podacima bez pisanja i pokretanja SQL naredbi. Django dolazi i s korisničkim sučeljem za obradu podataka pod nazivom Django Admin, koja omogućava četiri osnovne CRUD operacije nad svakom tablicom, kao i filtriranje skupa podataka. Primjer korištenja navedenog sučelja prikazan je na slici 7.



Slika 7 – Administratorsko sučelje u Djangu

Na ovaj način nije moguće raditi kompleksnije upite, iako to nije ni svrha admina. On služi jednostavnom dodavanju i uklanjanju podataka od strane krajnjih korisnika aplikacije, bez da moraju ulaziti u specifičnosti SQL ili Django okoline.

4.1. Pristup podacima iz aplikacije

Programski jezici uglavnom dolaze s osnovnom podrškom za pristup bazi podataka. U objektno-orijentiranim jezicima ugrađene su klase koje omogućuju spajanje na bazu podataka, izvršavanje naredbi, dohvaćanje podataka, rad s transakcijama i sl. Razvojni inženjer pomoću navedenih klasa razvija vlastite klase i komponente specifično za bazu podataka koja se koristi u određenoj aplikaciji. Za svaku tablicu u bazi uglavnom je potrebno napisati posebnu klasu s barem jednom metodom za umetanje, brisanje, ažuriranje i dohvaćanje podataka. Aplikacija može zahtijevati i različita filtriranja, sortiranja i istovremeno dohvaćanje podataka iz više tablica. Pisanje vlastitog koda za pristup bazi zahtijeva iznimno puno vremena, pogotovo ako se radi o bazi koja ima velik broj tablica. Osim toga, baza podataka se s vremenom mijenja, zbog promjene postojeće ili dodavanja nove funkcionalnosti. Navedene promjene se moraju

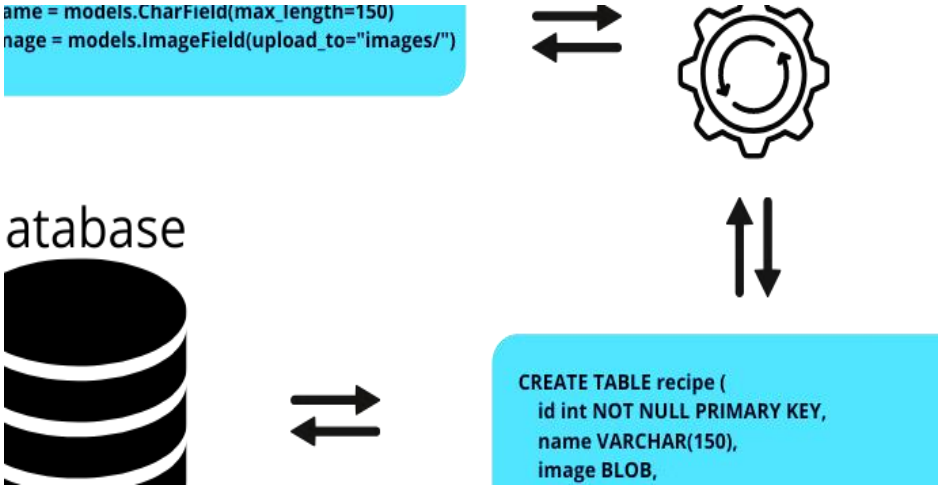
odraziti i na komponente u samoj aplikaciji, što zahtijeva dodatno vrijeme, a i povećava mogućnost pogrešaka.

Jedno od rješenja za smanjenje pisanja i održavanja velike količine koda su generatori koda, poput MyGeneration i CodeSmith. Navedeni alati omogućuju korisniku zadavanje skripte pomoću koje se definira predložak za tablicu, pogled ili proceduru iz baze podataka. Nakon pokretanja skripte se za svaku tablicu generiraju posebne klase, metode i sve ostalo što je korisnik definirao u skripti. Generirani kod je zatim potrebno uključiti u projekt, čime se omogućuje pristup novim objektima u bazi.

4.2. Objektno relacijsko mapiranje

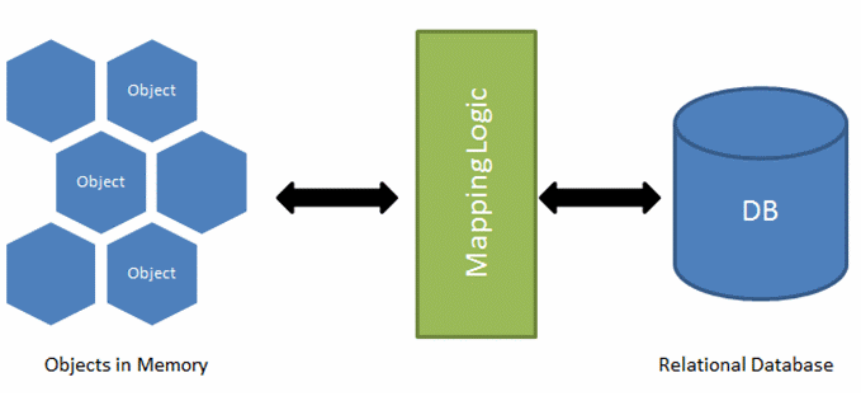
Objektno relacijsko mapiranje (ORM) predstavlja koncept pretvaranja koda napisanog objektno orijentiranom paradigmom u jezik relacijske baze podataka. To je metoda pretvaranja podataka u oblik s kojim može raditi aplikacija, odnosno neki sustav. Pomoću ORM sustava stvara se virtualno objektna baza prilagođena korištenju unutar programskog jezika (Roebuck, 2011).

Prvi korak u bilo kojem ORM okruženju uključuje definiranje budućih tablica u bazi. Tablicu opisujemo kao klasu, a nakon toga će ORM dobiveni kod prevesti ili mapirati u SQL naredbu, odnosno u ovom slučaju u kod kojim se u bazi kreira nova tablica. Slika 8. prikazuje navedenu funkcionalnost, odnosno generiranje koda za kreiranje tablice *recipe* na temelju klase *Recipe*. Vidljivo je da su se svojstva *name* i *image* klase mapirali u kolone tablice, ali je i dodana kolona *id*, koja predstavlja primarni ključ tablice.



Slika 8 - ORM konverzija koda u SQL tablice (Brian, 2021)

Osim mapiranja strukture, odnosno tablica i klasa, ORM omogućuje i mapiranje samih podataka. Jedan redak iz tablice u bazi podataka mapira se u jedan objekt u aplikaciji. Objekti postoje u memoriji aplikacija, a retci su zapisani u datoteci baze podataka, kao što je prikazano na slici 9. ORM ne omogućuje samo pristup podacima u bazi, već i njihovo brisanje, promjenu i umetanje. Ako se u aplikaciji obriše objekt, ORM omoguće da se navedena promjena sinkronizira s bazom podataka, odnosno omogućuje izvršavanje SQL DELETE naredbe s određenim identifikatorom. Analogno vrijedi za promjenu i kreiranje objekata, koji će se s bazom podataka sinkronizirati pomoću odgovarajućih UPDATE i INSERT naredbi.



Slika 9 - ORM mapiranje objekata i tablice (Shapavalov, 2021)

5. Django ORM

Django je razvojni okvir (engl. *framework*) unutar programskog jezika Python koji se sastoji od svih alata potrebnih za kreiranje web aplikacije. Može ga se promatrati kao implementaciju ORM koncepta. Koristi se kako bi se u programskom jeziku Python olakšala izrada aplikacije jer sadrži već gotove segmente aplikacije, odnosno modele, administratorsko sučelje, prikaz URL putanji itd.

Programiranje web aplikacija često uključuje ponavljanje istih zadataka. Takvi zadaci uključuju izradu administracijskog sučelja, autentikaciju korisnika, korisnička prava, izradu formulara za unos podataka, validaciju unesenih podataka, internacionalizaciju web aplikacija. Django okruženje služi obavljanju svih tih zadataka pisanjem svega nekoliko linija koda, bez da se pritom potrebno baviti jezikom baze jer ga Django automatski generira iz koda pisanog u Pythonu. Također, Django pruža vlastiti *shell* za interaktivno programiranje koji služi za brze promjene i analizu aplikacije.

U nastavku su prikazane najvažnije mogućnosti Django ORM-a.

5.1. Filtriranje baze podataka

Izdvajanje podataka iz baze prema upisanim kriterijima naziva se filtriranjem baze podataka. To je mogućnost, odnosno alat koji se u bazama podataka koristi kako bi se iz cijelog skupa podataka odabrali oni retci koji zadovoljavaju određene uvjete. Pri filtriranju podataka, svaki se novopostavljeni kriterij dodaje postojećim kriterijima, čime se olakšava rad s podacima. Primjer filtriranja podataka iz tablice User prikazan je na slici 10.

```
>>> from django.contrib.auth.models import User
>>>
>>> from polls.models import Question
>>>
>>>
>>> User.objects.all()
<QuerySet [<User: testuser>, <User: randomuser>]>
>>>
>>> User.objects.filter(is_staff=True)
<QuerySet [<User: testuser>]>
>>>
```

Slika 10 - Filtriranje podataka unutar Djanga

Navedeni kriteriji, tzv. *querysetovi* mogu se nizati jedan na drugog, čime se lakše konstruiraju kompleksni upiti ili je upite čak moguće i spojiti. Django optimizira vrijeme izvršavanja takvih upita na način da se queryset evaluira samo jednom, čime se omogućava dodavanje nekoliko filtera. Time se izbjegava direktno izvršavanje naredbi i filtriranja, to jest izbjegava se nepotrebno izvršavanje jednakih, odnosno dvostrukih upita nad bazom. Na slici 11 prikazan je primjer spajanja dva upita. U prvom upitu se zahtijeva da se vrate pitanja koja u tekstu sadrže znakovni niz *first*, a u drugom ona koja su upravo objavljena.

```
>>> qs = Question.objects.filter(question_text__icontains='First')
>>> qs.filter(pub_date__lte=timezone.now())
<QuerySet [<Question: First question?>]>
>>>
```

Slika 11 - Spajanje upita u Djangu

Ako je potrebna daljnja optimizacija kompleksnih upita, mogu se koristiti *select_related* ili *prefetch_related* funkcije, koje dohvaćaju sve objekte vezane uz objekte dobivene iz baze. Na taj način izbjegavaju se novi upiti nad bazom u slučaju dohvaćanja dodatnih podataka iz dobivenih rezultata. Na slici 12 prikazan je upit koji dohvaća sve podatke iz tablice Choice, ali i podatke iz tablice Question, s kojim je tablica povezana. Prvi redak iz prve tablice je *Choice: A*, a kad se želi ispisati tekst pitanja kojem pripada navedeni odabir (*First question*), nije potrebno raditi novi upit na bazu jer se podaci već nalaze u querysetu.

```
>>> Choice.objects.prefetch_related('question')
<QuerySet [<Choice: A>, <Choice: C>, <Choice: 1>, <Choice: 2>, <Choice: 3>, <Cho
ice: I>, <Choice: II>, <Choice: III>, <Choice: a3>, <Choice: 93>, <Choice: 92>,
<Choice: 91>, <Choice: a2>, <Choice: a1>, <Choice: B>]>
>>> qs = Choice.objects.prefetch_related('question')
>>> qs
<QuerySet [<Choice: A>, <Choice: C>, <Choice: 1>, <Choice: 2>, <Choice: 3>, <Cho
ice: I>, <Choice: II>, <Choice: III>, <Choice: a3>, <Choice: 93>, <Choice: 92>,
<Choice: 91>, <Choice: a2>, <Choice: a1>, <Choice: B>]>
>>> qs.first()
<Choice: A>
>>> qs.first().question
<Question: First question?>
>>> qs.first().question.question_text
'First question?'
>>>
```

Slika 12 - Dohvaćanje vezanih podataka sa prefetch_related funkcijom

5.2. Modeli

Srž Django ORM-a leži u modelima i svaki razvoj unutar ovog okruženja počinje s definiranjem modela. Model u obliku klase definira sve atribute, njihove tipove podataka te pravila validacije. Modelu u objektno orijentiranom jeziku odgovara tablica u bazi podataka. Samo ime klase je ime tablice, dok atributi predstavljaju stupce.

```
import datetime

from django.db import models

from django.utils import timezone

from django.contrib import admin


class Question(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)

    def __str__(self):
        return self.choice_text

    @property
    def text_size(self):
        return len(self.choice_text)

    @admin.display(
        boolean=True,
        ordering='pub_date',
        description='Published recently'
    )
    def text_size(self):
        return len(self.choice_text)
```

Kod 1 - Primjer modela, njegovih atributa i metoda

Koristeći django model aspekt, stvara se Python reprezentaciju SQL Questions tablice, koja ima jedno tekstualno polje *choice_text*, jedno numeričko polje *votes* te poveznicu stranim ključem na *question* tablicu.

Osim glavnih atributa, modeli u sebi sadrže i metode koje mogu dinamično manipulirati podacima u Pythonu te vratiti dinamične vrijednosti koje se inače ne mogu dobiti iz baze, bez vršenja dodatnih upita unutar SQL-a.

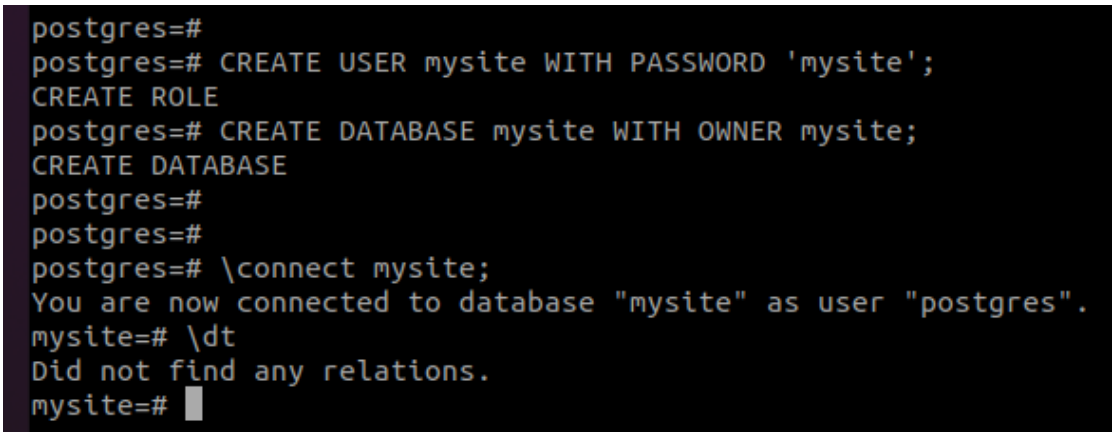
Druga važna značajka modela su *property metode*. One predstavljaju spoj metoda i atributa. Property metodu pozivamo kao bilo koji atribut, ali se u pozadini ona ponaša kao metoda. Uvijek vraća jednu vrijednost kao i atribut, ali pritom izbjegava upite nad bazom te direktno izračunava dinamičnu vrijednost u Pythonu. Na taj način se mogu dobiti informacije o duljini teksta putem *text_size* metode koja inače ne bi nužno bila dostupna putem čistog SQL-a.

```
class Question(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)

    def __str__(self):
        return self.choice_text

    @property
    def text_size(self):
        return len(self.choice_text)
```

Kod 2 - Django model property metoda

5.3. Migracije

Nakon definiranja modela, vrši se preslikavanje u sam SQL. Prvi korak uključuje ručno kreiranje baze unutar PostgreSQL-a. Slika 13 prikazuje primjer kreiranja baze podataka *mysite*, kao i korisnika koji joj može pristupiti.



Slika 13 - Stvaranje baze u PostgreSQL-u

Time je stvorena prazna baza podataka. Pomoću *CREATE TABLE* naredbi moguće je ručno stvaranje tablica, kako bi odgovarali prethodno kreiranim modelima, ali Django ORM automatizira taj proces. Nakon stvaranja baze, prvi korak unutar Djanga uključuje konfiguraciju pristupnih podataka baze. Taj dio se definira unutar *settings.py modula*, kao što je prikazano u kodu 3.

```
# settings.py
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'mysite',
        'USER': 'mysite',
        'PASSWORD': 'mysite',
        'HOST': 'localhost',
        'PORT': 5432,
    }
}
```

Kod 3 - Defniranje baze unutar Django postavki

Naredba pomoću koje se Python *shellom* automatski stvara tablica unutar baze podataka na temelju kreiranih modela je *manage.py migrate*. Migracije su čitljivi dokumenti na disku u koje Django sprema izmjene na modelima. Služe sinkronizaciji tablice s trenutnim stanjem unutar modela. Slika 14 prikazuje izlaz koji se prikazuje nakon pokretanja operacije migracije. U ovom

slučaju, baza podataka ne sadrži potrebne tablice, tako da će se one kreirati i baza će se uskladiti sa zadnjim promjenama u modelu.

```
(venv) sasa@sasa-HP-ENVY-x360-Convertible-13-ay0xxx:~/Desktop/mysite$ ./manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
```

Slika 14 - Stvaranje SQL tablica pomoću migracija

Provjeravanjem stanja baze unutar PostgreSQL-a, vidljivo je da je Django kreirao sve tablice po uzoru na modele, kao što je vidljivo na slici 15.

```
mysite=# \dt
                List of relations
 Schema |            Name            | Type  | Owner
--------+----------------------------+-------+--------
 public | auth_group                 | table | mysite
 public | auth_group_permissions     | table | mysite
 public | auth_permission            | table | mysite
 public | auth_user                  | table | mysite
 public | auth_user_groups           | table | mysite
 public | auth_user_user_permissions | table | mysite
 public | django_admin_log           | table | mysite
 public | django_content_type        | table | mysite
 public | django_migrations          | table | mysite
 public | django_session             | table | mysite
 public | polls_choice               | table | mysite
 public | polls_question             | table | mysite
(12 rows)
```

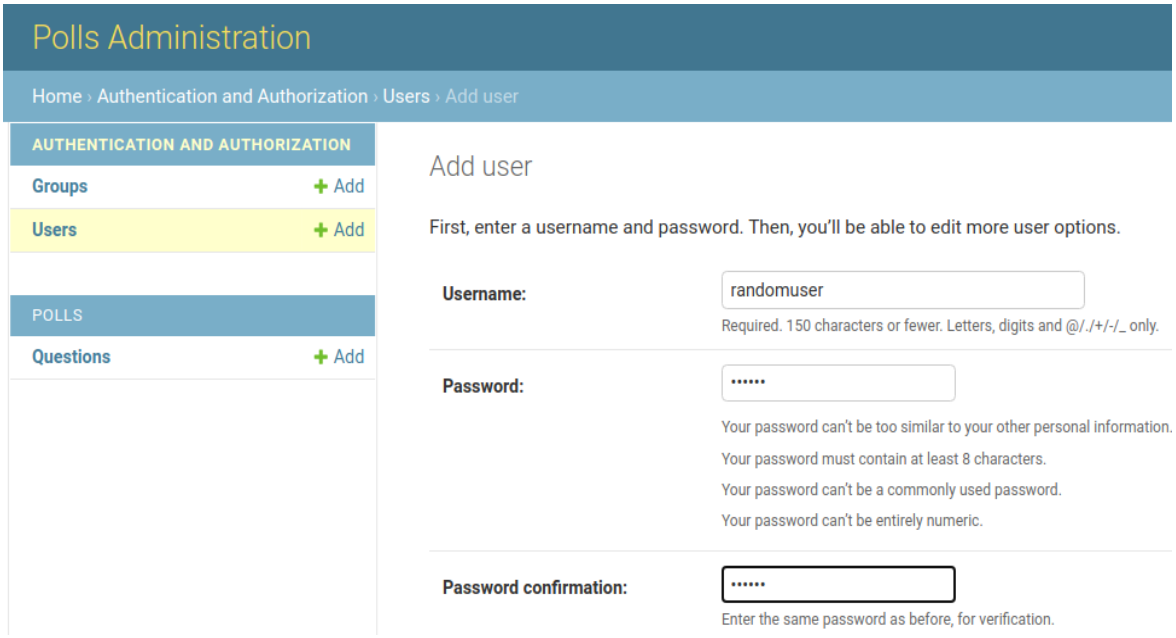
Slika 15 - Tablice u PostgreSQL bazi

Naravno, navedena naredba se također koristi za daljnje izmjene nad shemom baze. Svaka nova izmjena nad bazom se zapisuje kao *migration.py* datoteka. Migracije zajedno služe kao povijesni zapis svih promjena. Te datoteke moraju biti točne i posložene pravim redoslijedom s obzirom na potrebu čestog vraćanja izmjena nad bazom. Time se osigurava mogućnost promjene nad bazom u oba smjera (nove izmjene ili izmjena starih). Očuvanjem točnog slijeda migracija osigurava se sinkronizirano stanje između modela u Pythonu i tablica u bazi podataka.

5.4. Administratorsko sučelje

Jedan od glavnih alata Django okoline je administratorsko sučelje, čija je namjena obavljanje osnovnih CRUD operacija i filtiranje nad bazom za krajnje korisnike. Definira se kao unaprijed izgrađena aplikacija koja iz modela stvara forme pomoću kojih je moguće jednostavno manipulirati podacima u bazi (Django admin site, 2021). Administratorsko sučelje je dostupno putem */admin* URL-a, ali pristup je moguć samo administratorima.

Administratori mogu upravljati drugim korisnicima, njihovim grupama i razinama pristupa. Na taj način mogu dodati druge administratore ili limitirati krajnjim korisnicima pristup određenim funkcionalnostima. Slika 16 prikazuje formu za dodavanje novog korisnika.



Slika 16 - Dodavanje novog korisnika

Autenticirani korisnici imaju pristup svim tablicama u bazi podataka te mogu mijenjati njihovu strukturu i podatke. Osim osnovnih formi i filtriranja, administratorsko sučelje moguće je jednostavno prilagoditi i proširiti u svrhu zadovoljavanja posebnih potreba krajnjih korisnika. Npr. nekoliko modela (tablica) može se organizirati u jedan skup formi kako bi olakšao unos podataka korisniku za veće logičke cjeline.

Prikaz liste instanci dolazi i s formama za pretraživanje i filtriranje. Ove funkcionalnosti se jednostavno dodaju unutar Django admin.py modula. S vremenom se količina podataka unutar baze povećava, što može otežati pretragu. Ovi alati nam omogućavaju precizno i brzo dohvaćanje specifičnih setova podataka bez ručnog pretraživanja velikih tablica.

5.5. Forme

Osim preddefiniranih formi u administratorskom sučelju, moguće je stvaranje i vlastite forme po istom ORM principu. U donjem primjeru, ORM na isti nacin kao i SQL tablice u modelima, stvara i kontakt formu u HTML-u. Python klasa ContactForm svoje atribute poput subject ili message preslikava u form polja, sa tipovima i validacijskim pravilima definiranim unutar poziva forms.Fields objekata.

```
from django import forms

class ContactForm(forms.Form):
    subject = forms.CharField(max_length=100)
    message = forms.CharField(widget=forms.Textarea)
    sender = forms.EmailField()
    cc_myself = forms.BooleanField(required=False)
```

Kod 4 - Django ručna forma

Forme slijede isti uzorak kao i modeli. Definirani kao klase, njihovi atributi u ovom slučaju predstavljaju polja forme, također sa svim validacijskim pravilima.

Umjesto prilagođenog oblika, formu je moguće stvoriti izravno iz modela na način automatskog preslikavanja polja modela i njihovih validacijskih pravila u samu formu. Uzvevši navedeno pravilo u obzir, potrebno je pridržavati se DRY (engl. D*on't Repeat Yourself*) principa; ne ponavljati istu logiku u kodu.

```
# Create the form class.
>>> class ArticleForm(ModelForm):
...     class Meta:
...         model = Article
...         fields = ['pub_date', 'headline', 'content', 'reporter']
```

Kod 5 - Django model forma

Prilikom instanciranja forme u HTML-u, moguća je ručna izgradnja sa svim specifično definiranim poljima.

```
<form action="{% url 'polls:vote' question.id %}" method="post">
{% csrf_token %}
<fieldset>
    <legend><h1>{{ question.question_text }}</h1></legend>
    {% if error_message %}<p><strong>{{ error_message
}}</strong></p>{% endif %}
    {% for choice in question.choice_set.all %}
        <input type="radio" name="choice" id="choice{{
forloop.counter }}" value="{{ choice.id }}">
        <label for="choice{{ forloop.counter }}">{{
choice.choice_text }}</label><br>
    {% endfor %}
</fieldset>
<input type="submit" value="Vote">
</form>
```

Kod 6 - Django HTML ručna forma

Jednostavniji način uključuje definiranje samoizgrađene forme, koja na istom principu kao i prikazanom u kodu 7 automatski povlači i definira sva polja u HTML-u iz svoje forme definirane u Djangu.

```
<form action="/your-name/" method="post">
    {% csrf_token %}
    {{ form }}
    <input type="submit" value="Submit">
</form>
```

Kod 7 - Django HTML model forma

Spajanjem formi, modela i HTML formi s tim istim Python klasama, omogućava se jednostavan, brz i točan protok podataka između korisnika koji unosi informaciju u HTML formu, sve do baze koja zapisuje te podatke. Time se izbjegavaju izazovi zaboravljenih

validacijskih pravila, točnosti unesenih podataka ili hakerskih napada najčešće povezanih uz forme. U konkretnom slučaju, Django dolazi sa specifičnom zaštitom protiv CSRF (engl. *Cross-Site Request Forgery*) napada.

5.6. Serializatori

```
from rest_framework import serializers

class CommentSerializer(serializers.Serializer):
    email = serializers.EmailField()
    content = serializers.CharField(max_length=200)
    created = serializers.DateTimeField()
```

Kod 8 - Django serializator

Serializatori su ekvivalenti formi, specifično dizajnirani za API (engl. *Application Programming Interface*), razrađen dalje u tekstu. Funkcioniraju na istom principu koji uključuje preslikavanje cjelokupne logike iz modela na serializer klasu. Serializator na isti način vrši validaciju svakog polja.

Serializatori i forme funkcionalno su isti. Forme služe za unos podataka od strane korisnika, dok serializatori obrađuju podatke koji dolaze s određenog servisa ili aplikacije. Serializator klasa proizlazi iz modela i na temelju njegovih atributa validira unesene podatke. Dobivene rezultate formatira na specifičan način.

```
serializer = CommentSerializer(comment)
serializer.data
# {'email': 'leila@example.com', 'content': 'foo bar', 'created': '2016-01-27T15:17:10
.375877'}
```

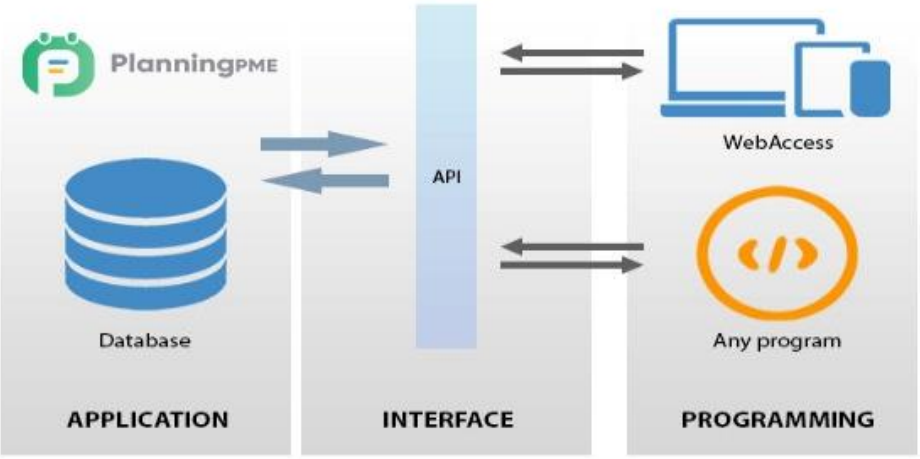
Kod 9 - Izlaz serializatora u JSON formatu

Forme strukturiraju podatke u HTML-u, dok serializatori čine istu stvar u JSON-a ili XML-u. Serializatori se najčešće koriste u radu s API aplikacijama. U navedenom, oni zajedno s Django

modelima predstavljaju model MVC (engl. *model-view-controller*) arhitekture. Osim API-ja, serializatori se mogu zasebno koristiti unutar manjih operacija, skripti ili automatskog unosa/ispisa podataka. U tom kontekstu, služe samo kao validacijska ekstenzija modela (*Django REST Framework, 2021.)*

5.7. API

API ili *Application Programming Interface* je koncept koji omogućava međusobnu komunikaciju između aplikacija ili servisa. Drugim riječima, funkcionalnost aplikacije pružena krajnjem korisniku preslikana je u API format, na način da se iste akcije mogu izvršiti iz neke druge aplikacije ili sustava, kao što je prikazano na slici 18.



Slika 17 - API arhitektura (PlanningPME, 2021)

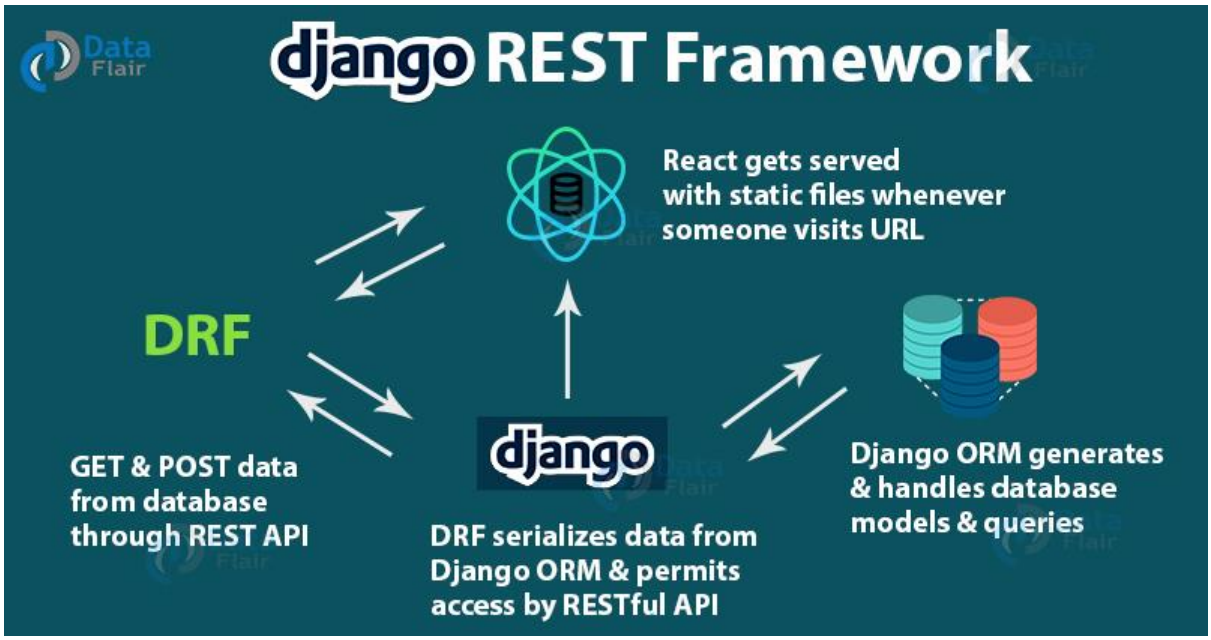
Uzevši Youtube platformu kao primjer, krajnjem korisniku je omogućeno ručno kreiranje *playlista*, dodavanje novih pjesama i mijenjanje redoslijeda istih. Korisnik također može izraditi aplikaciju koja će se spojiti na Youtube API te na automatizirani način izvršiti sve prethodno navedene radnje unutar par sekundi.

Većina današnjih aplikacija koristi druge API-je kako bi nadogradili svoju funkcionalnost. Štoviše, većina aplikacija su i same konstruirane kao API servisi, budući da jedan server najčešće mora servirati više klijenata (browser, mobile, microservisi).

DRF ili *Django REST Framework* predstavlja Django ekstenziju koja omogućava izgradnju API aplikacija. REST ili reprezentativni prijenos stanja je princip koji bi trebao biti prisutan u svakoj API aplikaciji. Omogućava API servisima vršenje asinkronih zahtjeva. Uz pomoć spomenutog alata, servis koji koristi API servis ne mora brinuti o prethodnim akcijama, duljini

korisničke sesije ili drugim limitacijama. Navedeni benefiti rezultiraju fleksibilnijom i sinkronijom razmjenom podataka (Django REST Framework).

Sama implementacija takvog sustava u Djangu vrlo je jednostavna. Pridržavajući se Djangovog MVC modela, vidljivo je da je on neizmjenjen, osim što sada modeli i serializatori funkcioniraju kao jedna cjelina, a ispis podataka više nije u HTML formatu koji se renderira krajnjem korisniku već je on unutar JSON ili XML formata. Umjesto renderiranja, navedeni podaci šalju se drugom servisu radi daljnje obrade podataka na toj platformi.



Slika 18 – Django REST Framework arhitektura (DataFlair, 2019)

Na slici 19 vidljivo je moguća arhitektura u slučaju korištenja SPA (engl. S*ingle-Page Application*) aplikacije, koja dinamično ažurira postojeću web stranicu za razliku od onih koje učitavaju cijelu web stranicu po svakom zahtjevu. Veza između baze i modela ostaje ista, dok se umjesto formi za validaciju koriste serializatori.

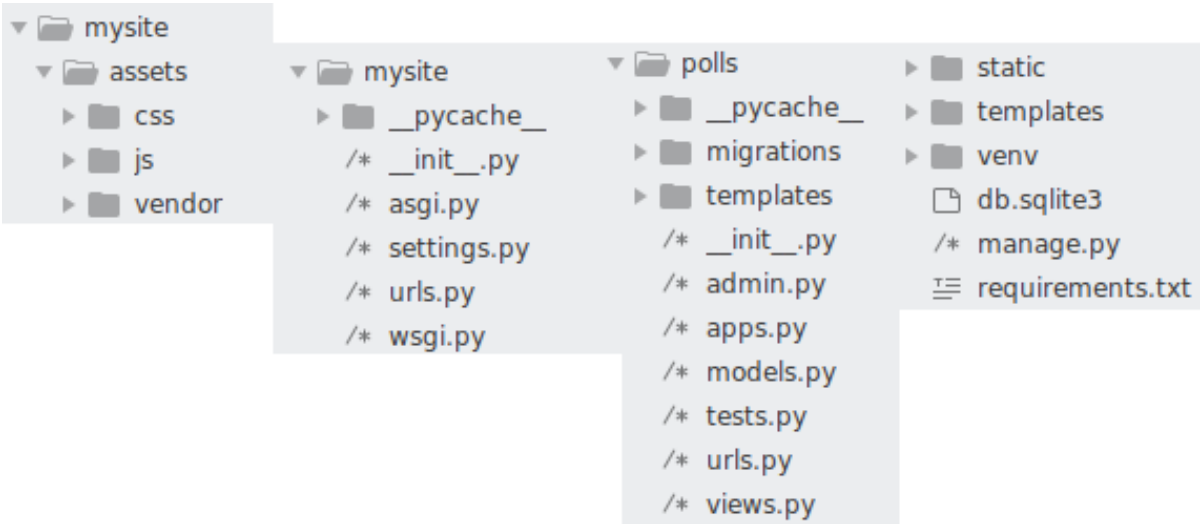
Kontroler aspekt je još uvijek *urls.py* modul, ali njegove putanje su mnogo striktnije. Za početak, ne prihvaćaju HTML sadržaj, što onemogućava korištenje API kao obične Django aplikacije od strane krajnjeg korisnika. Umjesto toga, renderiranje i vizualna reprezentacija podataka prikazuju se u Javascript ili mobilnoj aplikaciji koja sama konstruira svoje URL putanje te ih na dinamičan način povezuje s API putanjama.

6. Web aplikacija

U sklopu rada izrađena je web aplikacija u programskom jeziku Python, koja pomoću Django ORM-a pristupa podacima u PostgreSQL bazi podataka. U ovom poglavlju opisana je struktura projekta, zajedno s direktorijima i najvažnijim datotekama, te je opisan izgled i funkcionalnost korisničkog i administratorskog dijela aplikacije.

6.1. Struktura web projekta

Koristeći naredbu *django-admin startproject mysite*, Django samostalno generira sve datoteke i sadržaj potreban za inicijalni razvoj aplikacije. Struktura projekta je prikazana na slici 20.



Slika 19 - Struktura projekta

Assets

Naziv označava direktorij za sve statične datoteke, poput CSS i Javascript resursa. To su najčešće datoteke preuzete iz drugih izvora, poput Bootstrap CSS razvojnog okvira.

Mysite

Mysite se definira kao projektni direktorij koji pohranjuje sve postavke Django aplikacije i najčešće je istog naziva kao i glavni direktorij. *asgi.py i wsgi.py* su ekstenzije koje se spajaju na web server koji će posluživati samu aplikaciju. Konkretno, kod izrađene aplikacije je web server NGINX.

Settings.py je glavni modul u kojem su konfigurirane sve komponente, predefinirane od strane Djanga. U *settings.py* modulu postavljaju se podaci za pristup bazi, naziv domene, putanje za statičke i HTML datoteke te definicije integracija drugih paketa.

Urls.py je kontroler komponenta MVC sustava koja spaja URL putanje s logičkim cjelinama (view-ima) zaduženim za obradu, odnosno dohvaća podatke iz baze te vraća rezultate u HTML ili JSON formatu.

Polls

Polls predstavlja zasebnu aplikaciju unutar *mysite* projekta koja je također generirana s Django naredbom *python manage.py startapp polls*. Uz njezinu pomoć, odrađuje se kreiranje svih potrebnih direktorija i datoteka. Srž logike nalazi se u *models.py*, gdje se definiraju modeli, to jest njihov ekvivalent tablica u SQL-u. Spomenute modele koristi *views.py* koji ih poziva koristeći ORM te putem *queryset* filtriranja dohvaća željene podatke za specifičnu URL rutu. Konačno, *urls.py* jednako kao i u projektnom direktoriju povezuje rute sa zadanim URL slijedovima.

Static

Poput projektnog assets direktorija, static služi kao glavno spremište svih statičnih datoteka za prikaz HTML-a. S obzirom na mogućnost višestrukih izvora ovakvih datoteka, Django ih naknadno grupira na jedno mjesto s ciljem dostupnosti svih datoteka serveru.

Templates

Django koristi Jinja *template* jezik; običan HTML s elementima OOP-a. Na taj način omogućeno je strukturiranje HTML datoteke u komponente te programsko ubacivanje podataka kao dinamične HTML vrijednosti. Nerijetko nazvane i kontekstne varijable, popunjavaju mjesto u HTML-u s podacima koje Django mora zasebno procesuirati po svakom zahtjevu. Kao i *static* direktorij, sve HTML datoteke najčešće se nalaze jednom mjestu.

VENV

VENV (engl. *Virtual ENVironment*) je u doslovnom prijevodu virtualno okruženje. Svaka aplikacija posjeduje drugačiju specifikaciju paketa, drugačije jezike, verzije jezika te drugačije tehnologije, poput razvojnih okvira ili baza podataka. U svrhu izbjegavanja poteškoća u kontekstu kompatibilnosti sustava i aplikacije, koriste se metode virtualizacije kako bi imali čistu, izoliranu okolinu na čijoj podlozi može funkcionirati aplikacija kreirana u radu.

Db.sqlite3

Sqlite je mala, jednostavna baza podataka, koja se zapisuje izravno u direktorij projekta kao tekstualna datoteka. U izradi aplikacije, lokalno je korišten PostgreSQL kao precizniji prikaz stvaranja baze, međutim ipak je odabrana jednostavnija opcija zbog ograničenja PythonAnywhere cloud platforme.

Manage.py

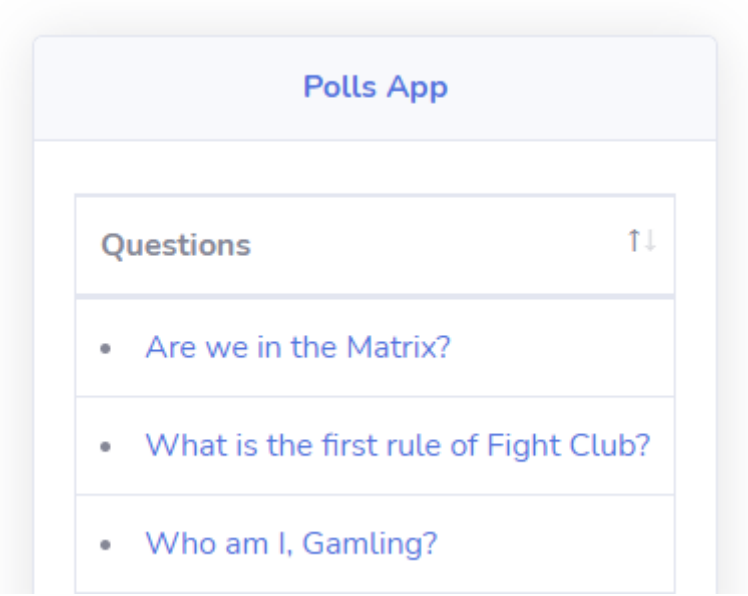
Manage.py predstavlja glavni razvojni alat Django okruženja. Modul omogućava pokretanje svih naredbi prilikom razvoja i postavljanja aplikacije. Služi za dizanje servera, pokretanje *interactive shell* alata i zasebnih naredbi te u konačnici generalno stvaranje projekta, aplikacija i dohvaćanje statičnih datoteka.

Requirements.txt

Requirements.txt uključuje popis paketa i njihovih verzija koje koristimo prilikom pokretanja aplikacije. Ova datoteka mora biti točna i često ažurirana s obzirom na učestalu promjenu paketa i njihovu tendenciju postajanja nekompatibilnima**.**

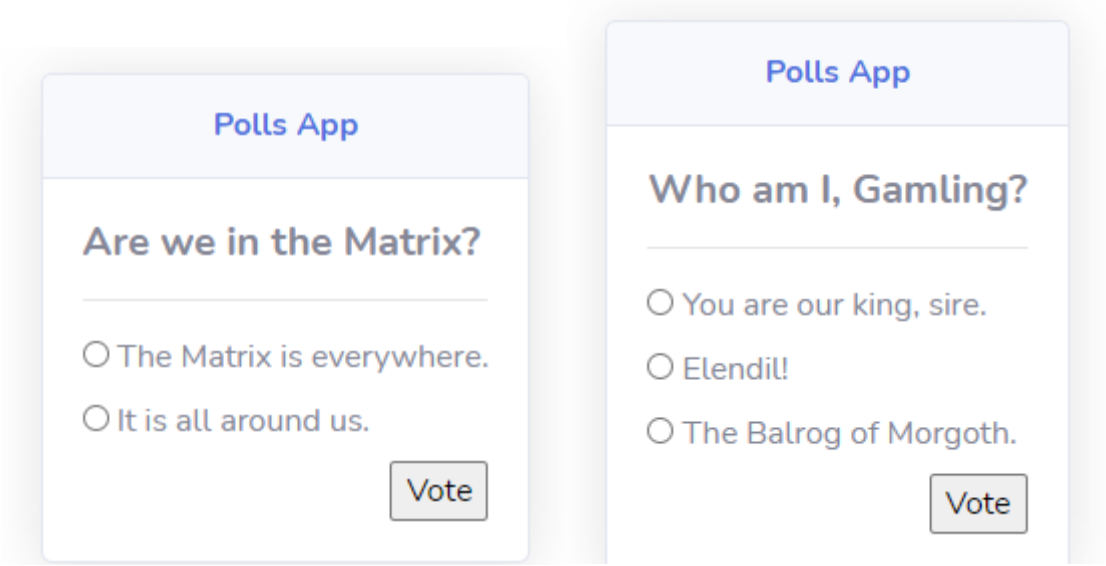
6.2. Funkcionalnost i izgled aplikacije

Sama logika ankete vrlo je jednostavna. Za početak, na slici 21 vidljiv je popis svih trenutnih anketa, odnosno pitanja. To je *landing page* koju korisnik vidi čim posjeti domenu. Lista pitanja sadrži poveznice na druge dijelove aplikacije, poput detalja pitanja.



Slika 20 - Lista pitanja

Na drugom koraku otvorena je mogućnost glasanja odgovorom na postavljeno pitanje nakon čega trenutni rezultati ankete postaju vidljivim. Glasanje se omogućava uz pomoć jednostavne formom koja sadrži polja sa *submit* akcijom na kraju. Samo jedna opcija može biti odabrana.

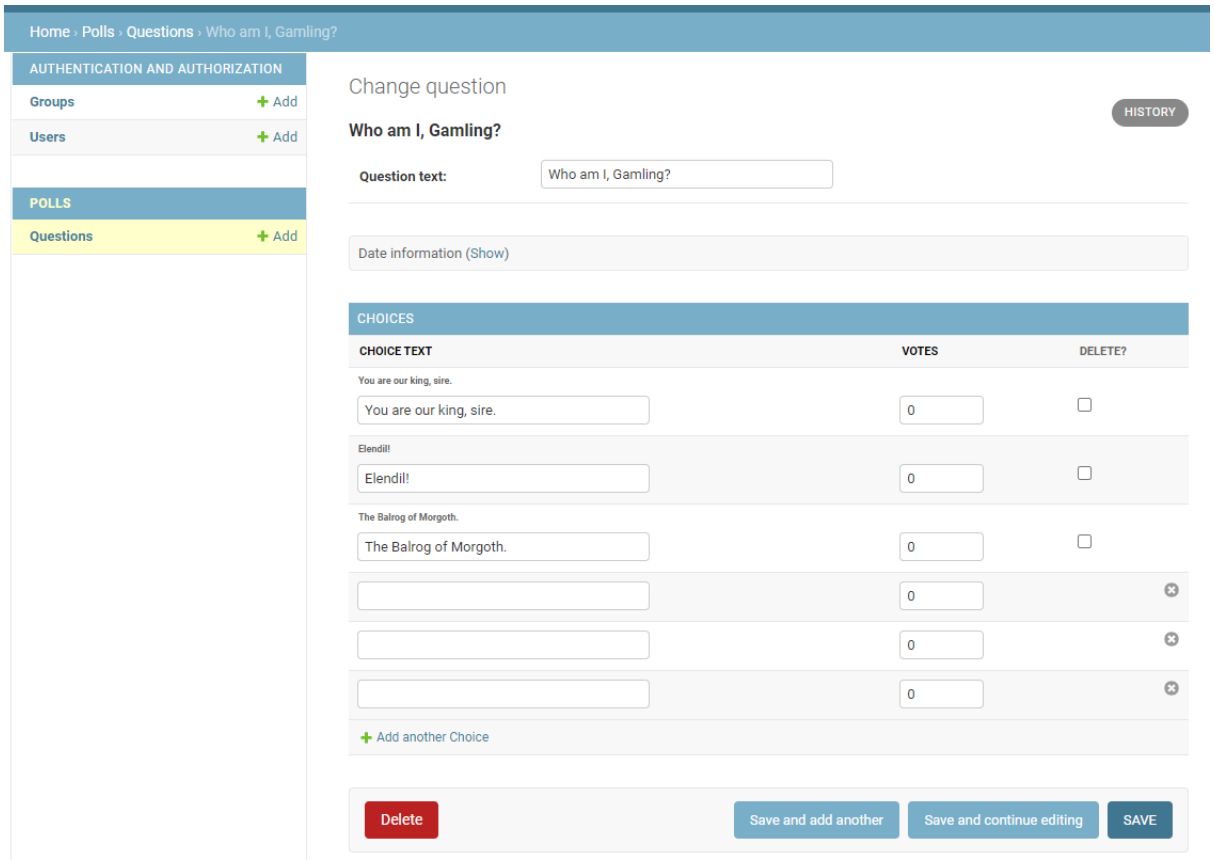


Slika 21 - Lista odgovora i rezultata

Administratori imaju mogućnost kontrole nad pitanjima, odgovorima i drugim korisnicima uz pomoć Djangovog admin sučelja, specifično dizajniranog za jednostavnu uporabu. Odlaskom na subdomenu */admin* korisnik se može autenticirati kao administrator. Nakon uspješne autentikacije, prisutan je prikaz svih tablica i njihovih instanci. Svaka tablica sadrži preglede za

uređivanje, dodavanje, brisanje i filtriranje. Klikom na tablicu, otvara se lista svih njenih instanci. U gornjem lijevom kutu vidljive su *batch* akcije, dok su u desnom alati za filtriranje.

Klikom na jednu od tih instanci, ulazi se u pregled za uređivanje gdje je moguća promjena sadržaja instance.



Slika 22 – Sučelje za unos podataka

7. Zaključak

Relacijske baze podataka osiguravaju neovisnost podataka i strukturalnu jednostavnost, što ih čini široko rasprostranjenim i opće prihvaćenim modelom strukturiranja podataka. U ovom radu pobliže su objašnjene, klasificirane te je prikazan primjer njihovog korištenja u web aplikaciji.

Objektno-relacijsko mapiranje (ORM) i Django imaju brojne prednosti od kojih se najviše ističu brzina razvoja i održavanja, posebice u primjeni na velikim projektima, te fleksibilnost koda koji omogućava jednostavnost korištenja.

Aplikacija razvijena za potrebe ovog rada pruža jednostavno organiziranje upita i glasova u obliku ankete. Na ovom primjeru vidljive su brojne prednosti Django ORM-a, kao što su stvaranje tablice, postavljanje interaktivnog korisničkog sučelja, kako za krajnje korisnike, tako i za administratore. Korištenjem naprednijih alata, poput stvaranja formi preko istog model principa, omogućena je integracija aplikacije s vanjskim servisima, poput drugih servera, mobilnih i SPA aplikacija. Aplikacija je postavljena na mrežu putem Python Anywhere cloud platforme, čime je omogućen stvarni pristup krajnjim korisnicima.

Literatura

1. Ardeleanu, S. 2016. Relational Database Programming: A Set-Oriented Approach, 1st Edition. Apress.
2. Berg, K.L., Seymour, T., Goel, R. 2012. History Of Databases. International Journal of Management & Information Systems (IJMIS), 17(1), pp.29–36.
3. Bouhsine, T. 2020. Design And Full Stack Development Of A Crowdfunding Platform «Sahem». Dostupno na: https://www.researchgate.net/publication/343988849_Design_And_Full_Stack_Development_Of_A_Crowdfunding_Platform_Sahem (24.6.2021.)
4. Brian. 2021. Introduction to the Django ORM. Engineer To Developer. Dostupno na: https://engineertodeveloper.com/introduction-to-the-django-orm (1.7.2021.)
5. Codd, E. F. 2002. A Relational Model of Data for Large Shared Data Banks. In: Software Pioneers. pp.263–294. Dostupno na: https://www.seas.upenn.edu/~zives/03f/cis550/codd.pdf (22.6.2021.)
6. DataFlair. 2019. Django REST Framework Tutorial. Dostupno na: https://data-flair.training/blogs/django-rest-framework (28.6.2021.)
7. Django admin site. 2021. Dostupno na: https://docs.djangoproject.com/en/3.2/ref/contrib/admin (28.6.2021.)
8. Django official documentation. 2021. Dostupno na: https://docs.djangoproject.com/en/3.2/ (1.7.2021.)
9. Django REST Framework. 2021. Dostupno na: https://www.django-rest-framework.org (29.6.2021.)
10. Greenfeld, D. R., Greenfeld, A. R. 2015. Two scoops of Django: Best practices for Django 1.8. Two Scoops Press.
11. Gyorödi, C., Gyorödi, R., Sotoc, R. 2015. A comparative study of relational and non-relational database models in a web-based application. *International Journal of Advanced Computer Science and Applications*, 6(11), 78-83.
12. Han, J., Haihong, E., Le, G., Du, J. 2011. Survey on NoSQL database. *In 2011 6th international conference on pervasive computing and applications* (pp. 363-366). IEEE.

13. Liu, S. 2021. Most popular relational database management systems 2021. *Statista*. Dostupno na: https://www.statista.com/statistics/1131568/worldwide-popularity-ranking-relational-database-management-systems/ (24.6.2021.)

14. Manger, R. 2003. Skripta iz Baza podataka. Dostupno na: http://jadran.izor.hr/~dadic/EKO/baze-podataka.pdf (20.6.2021.)

15. Microsoft.com. 2021. SQL Server 2019—Pricing | Microsoft. Dostupno na: https://www.microsoft.com/en-us/sql-server/sql-server-2019-pricing (23.6.2021.)

16. Moniruzzaman, A. B. M., Hossain, S. A. 2013. Nosql database: New era of databases for big data analytics-classification, characteristics and comparison. arXiv preprint arXiv:1307.0191.

17. Mysql.com. 2021. MySQL :: MySQL Editions. Dostupno na: https://www.mysql.com/products/ (22.6.2021.)

18. NGINX. 2021. How to Choose a Service Mesh. Dostupno na: https://www.nginx.com/blog/how-to-choose-a-service-mesh (1.7.2021.)

19. Oracle.com. 2021. Oracle Assets. Dostupno na: https://www.oracle.com/assets (22.6.2021.)

20. Oyom, A.N. 2017. Understanding the MVC pattern in Django. Medium. Dostupno na: https://medium.com/shecodeafrica/understanding-the-mvc-pattern-in-django-edda05b9f43f (25.6.2021.)

21. PlanningPME. 2021. Efficient and userfriendly scheduling solutions. Dostupno na: https://www.planningpme.com (25.6.2021.)

22. PostgreSQL.org. 2021. PostgreSQL: Versioning Policy. Dostupno na: https://www.postgresql.org/support/versioning/ (23.6.2021.)

23. PyPI. 2021. The Python Package Index. Dostupno na https://pypi.org (1.7.2021.)

24. PythonAnywhere. 2021. Host, run, and code Python in the cloud. Dostupno na: https://www.pythonanywhere.com (1.7.2021.)

25. Python.org. 2021. What is Python? Executive Summary. Dostupno na: https://www.python.org/doc/essays/blurb (1.7.2021.)

26. Ramakrishnan, R., Gehrke, J. 2011. Database Management Systems 2nd Edition. Dostupno na: https://xuanhien.files.wordpress.com/2011/04/database-management-systems-raghu-ramakrishnan.pdf (20.6.2021.)

27. Roebuck, K. 2011. Object-relational mapping (ORM): High-impact Strategies-What You Need to Know: Definitions, Adoptions. Impact, Benefits, Maturity, Vendors. Tebbo.

28. Shapavalov, A. 2021. *Micro ORM vs ORM*. Yaplex. Dostupno na: https://www.yaplex.com/blog/micro-orm-vs-orm (1.7.2021.)

29. Shaw, Z. A. 2018. Learn Python the Hard Way: Third Edition, Addison-Wesley.

30. Singh, V. 2021. What is Frameworks? [Definition] Types of Frameworks. Hackr.io. Dostupno na: https://hackr.io/blog/what-is-frameworks (1.7.2021.)

31. Yu, S. 2009. ACID Properties in Distributed Databases. *Advanced eBusiness Transactions for B2B-Collaborations*.

Popis slika

Popis kodova

Prilozi

Prilog 1 – Poveznica na aplikaciju i upute za njezino korištenje

Korisnički dio aplikacije je dostupan na adresi:
https://boldmarshmallow.pythonanywhere.com/

Administratorski dio aplikacije je dostupan na adresi:
https://boldmarshmallow.pythonanywhere.com/admin/

Pristupni podaci:

- Korisničko ime: lorenahorvat
- Lozinka: lorena

Sažetak

Objektno relacijsko mapiranje u Django web okolini

U radu se opisuju različiti pristupi relacijskim bazama podataka iz objektno-orijentiranih jezika. Detaljno se obrađuje objektno-relacijsko mapiranje (engl. *Object Relational Mapping, ORM*) te istražuju njegove prednosti i nedostaci. Opisani su najpopularniji ORM razvojni okviri i analizirane su njihove mogućnosti, pri čemu je poseban fokus na Django razvojnom okviru, njegovoj primjeni i integraciji u razvojni proces. U sklopu rada razvijena je aplikacija u programskom jeziku Python, koja koristi ORM za komunikaciju sa PostgreSQL bazom podataka. Osim jednostavnih operacija s podacima, demonstrirana su ažuriranja strukture tablica kroz migracije, načini korištenja ORM-a za obradu podataka u drugim formatima (JSON, XML) te definirani Python upiti koji se izvršavaju nad bazom podataka.

Ključne riječi: *objektno relacijsko mapiranje, relacijske baze, razvojni okviri, Django, izrada aplikacije, Python*

Summary

Object relational mapping in Django web framework

This paper focuses on different approaches to relational databases from object oriented programming languages. Object relational mapping (ORM) is discussed at lenght, and its strenghts and weaknesses are further explored. ORM frameworks are described with emphasis on their capabilities specifically focusing on Django framework, its application and integration into the developing process. Furthermore, an application programmed in Python programming language which uses ORM to communicate with PostgreSQL database is developed for the purposes of this paper. Alongside simple data operations, updating the structure of the tables, ways of using ORM to process data in other formates (JSON, XML) and defining Python queries executing over database are also demonstrated.

Key words: *object relational mapping, relational databases, frameworks, Django, programming, application, Python*