RAZVOJ APLIKACIJE ZA UPRAVLJANJE ZGRADOM KORIŠTENJEM RAZVOJNOGA OKVIRA DJANGO I POSTAVLJANJE NA VIRTUALNI SERVER

PRANIĆ, TINO

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijske tehnologije

TINO PRANIĆ

ZAVRŠNI RAD

RAZVOJ APLIKACIJE ZA UPRAVLJANJE ZGRADOM KORIŠTENJEM RAZVOJNOGA OKVIRA DJANGO I POSTAVLJANJE NA VIRTUALNI SERVER

Split, rujan 2020.

SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Preddiplomski stručni studij Informacijske tehnologije

Predmet: Informacijski sustavi

ZAVRŠNI RAD

Kandidat: Tino Pranić

Naslov rada: Razvoj aplikacije za upravljanje zgradom korištenjem razvojnoga okvira Django i postavljanje na virtualni server

Mentor: dr. sc. Igor Nazor, profesor visoke škole

Split, rujan 2020.

Sadržaj

Sažetak

Ovaj završni rad opisuje izradu aplikacije u Djangu, razvojnom okruženju zasnovanom na programskom jeziku Python. U prvom poglavlju rada je opisana metodologija razvoja informacijskoga sustava koji je nužan za daljnji razvoj aplikacije. U drugom poglavlju je opisan razvojni okvir Django, a u trećem poglavlju postupak izrade i osnovne karakteristike aplikacije.

Aplikacija prikazuje sustav plaćanja računa koje korisnik može pregledavati i plaćati. Administrator izrađuje račune koje korisnik može vidjeti u svom sučelju. Sustav prikazuje i neke ostale osnovne funkcionalnosti za jednostavan rad.

Summary

Developing an application for building menagement by using Django web framework and uploading to the virtual server.

This bachelor thesis describes the development process of an application based on Django, web framework that uses Python programming language. The first chapter includes basic methodology of information system development which is necessary for further development of the application. In the second chapter is described Django web framework, and in third chapter process of developing and basic characteristics of an application.

Application shows the system of paying the bills which user can review and pay. Administrator makes the bills which user can see in his interface. System shows some others basic functionalities for simple work.

1.Uvod

Internet je svakim danom sve potrebniji alat u svakodnevnome životu, s tim je i razvijanje raznih aplikacija postalo traženije na tržištu. Danas je digitalizacija sve prisutnija u bilo kojem segmentu života. Plaćanje online računa je već uvedeno u opću primjenu u razvijenom društvu. Budućnost plaćanja se možda i već u bližoj budućnosti može dovesti na višu razinu.

Odlučio sam pisati o ovom radu da bi diskusijom i razradom doprinijeo digitalizaciji procesa upravljanja zgradom razvojem i implementiranjem aplikacije korištenjem Django razvojnoga okvira. Neke od funkcija aplikacije su prikazane preko metodologije za razvijanje informacijskoga sustava. Jedan dio obuhvaća korištene metode u izradi aplikacije i samo programiranje korištenjem razvojnog okruženja za web zasnovanog na programskom jeziku Python.

U drugom dijelu rada je detaljnije opisano razvojno okruženje Django, uz kratak opis njegovog nastanka i najčešćih primjena. Navedeni su preduvjeti za njegovo korištenje, postupak instalacije i osnovne konfiguracije, komande u naredbenom retku, korištene metode uz opis rješavanja konkretnih problema i ispise koda na serverskom dijelu aplikacije.

2. Razvoj informacijskog sustava

Kako je razvoj informacijskog sustava kompleksan projekt koji se sastoji od dokumentiranja korisnikovih potreba i dizajna arhitekture aplikacije, postoje različite metodologije koje u tome pomažu. U ovom radu je korišten postupak koji se sastoji od sljedećih glavnih koraka: razgovor s korisnikom, funkcionalna dekompozicija, prikaz tijeka odvijanja procesa (radni dijagram) i izrada modela podataka.

2.1. Razgovor s korisnikom

Za razvoj aplikacije najprije je potrebno utvrditi zahtjeve od strane korisnika. To je zadatak u početnom stadiju planiranja gdje je potrebno prikupiti detaljne informacije kao temelj za kvalitetan ustroj i razvoj informacijskog sustava.

Prvi korak je utvrđivanje ciljeva poslovanja gdje analitičar ili projektant pomoću strukturiranih intervjua utvrđuju koje ciljeve i kakve poslovne procese naručitelj zahtjeva za željeni projekt. Za svaki poslovni sustav nužan je profit a to bi za ovakav tip projekta bilo omogućeno kvalitetom krajnje izvedbe. Za naručitelja je nužno da se aplikacija učestalo ažurira, da može pratiti račune i dug stanara, također da po stanu može pratiti dug, a sve to uz jednostavno korištenje. Sve nabrojano se može svesti na kvalitetu, jednostavnost, brzinu i efikasnost zadanog sustava.

Za ovakav tip projekta bi bilo poželjno da analitičar postavi anketu unutar zgrade ili više njih koje su predodređene za implementaciju ovakvoga sustava. Takva anketa bi sadržavala upite za stanare što bi tražili od aplikacije preko koje bi mogli unutar sustava plaćati račune i koje bi ostale funkcionalnosti željeli da sadrži. Teško je navesti sve potrebe za savršenu funkcionalnost sustava no za početak mogu se zadati one nužne za njegov neophodan rad.

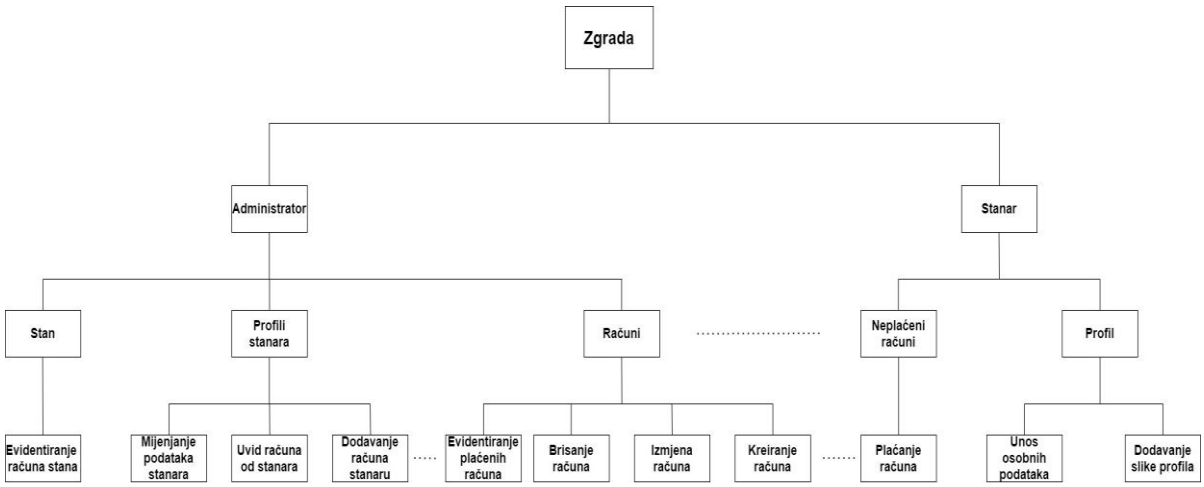
Sustav bi se temeljio pretežno na aplikacijskim rješenjima ali bi trebao i podržavati neka tehnička rješenja ako je takva potreba za digitalizaciju zgrade. Takva rješenja bi bila uvođenje terminala u nekome dijelu zgrade ili u samim stanovima. Ako je vanjski terminal trebao bi imati pojačano kućište i veću zaštitu, po mogućnosti s nadzornim kamerama.

Praktičnije rješenje bi bilo aplikacijsko. U tom slučaju sustav bi trebao sadržavati pojačanu zaštitu samoga *softwarea* radi utjecaja vanjskih čimbenika na aplikaciju. To znači podešeni *firewall* i antivirusni program s automatskom nadogradnjom i automatski backup podataka. Rješenje s terminalom ne treba imati registar korisnika dok bi aplikacijsko to sadržavalo. Aplikacija treba imati autorizaciju na login stranici i po mogućnosti da se svaki ulazak na stranicu zabilježava. Na stranici stanara treba postojati mogućnost za plaćanje računa kartično, putem PayPala, Stripea ili nekoga drugoga online servisa za plaćanje. Korisnička stranica treba sadržavati i korisnički profil u kojemu će korisnik moći mijenjati svoje osobne informacije preko korisničke unosne maske. Administratorska glavna stranica bi morala imati mogućnost kreiranja računa, praćenje plaćenih troškova te uvid koliko koji stan ima računa. Korisničko sučelje mora imati jednostavan i funkcionalan dizajn. Tako bi se korisniku pružio brz i jednostavan uvid u račune.

2.2. Dekompozicija funkcije

Pomoću dekompozicije funkcije sustav se razlaže na manje segmente ili podsustave. Najčešće se koriste dva načina za razlaganje, a to su *top-down* metoda tj. vertikalna metoda i vodoravna metoda. Vodoravna metoda glavni sustav prikazuje s lijeve strane i onda se grana na ostale funkcije. Vertikalna metoda se prikazuje tako da na vrhu stoji sustav.

Dekompozicija se vrši za informacijski sustav za upravljanje zgradom. Unutar aplikacije moguće je dodavanje više zgrada s toga bi ovakav projekt imao spojen sustav više njih. Primjer prikazuje jednu zgradu sa svojim funkcijama, procesima i procedurama. Sustav je potrebno razložiti na što manje segmente čiji se podsustavi ne mogu više razdvojiti na neke manje sustave koji bi se na kraju mogli prikazati jednim jednostavnim dijagramom. To znači da bi zadnji dijelovi odnosno procedure trebale biti konačne. Funkcije se detaljnije razlažu na dijagram toka podataka.



Slika 1. Dekompozicija funkcija sustava za upravljanje zgradom

Na Slici 1. je prikazana dekompozicija sustava koja se grana na dva podsustava Administrator i Stanar, koji se dalje granaju na podfunkcije.

Administrator sadrži glavnu funkciju za upravljanje daljnjim sustavom i raščlanjuje se na tri procesa: Stan, Profil stanara i Računi.

Proces Stan sadrži samo jednu proceduru koja je kreirana kao evidencija neplaćenih računa što znači da prikazuje dug stana. Ta procedura je jedino i važna za početnu funkciju željenoga sustava u vezi uključivanja stana u sustav.

U procesu Profil stanara administrator ima mogućnost mijenjanja bazičnih podataka o stanaru, kreiranje računa za određenoga stanara i evidenciju računa.

U nastavku je ukratko opisana tehnička izvedba pojedinih važnijih funkcionalnosti u aplikaciji Django.

Mijenjanje podataka stanara je u aplikaciji implementirano pomoću unosne maske jednostavnoga dizajna s osobnim podacima stanara koji su uključeni pomoću Django metode i mogu se promijeniti i potvrditi preko submit dugmeta.

Kreiranje računa je također ostvareno preko unosne maske. Rješenje je postavljeno na ovaj način radi dinamičnosti i jednostavnosti koje se mogu implementirati unutar sustava zbog brzine i efikasnosti koju pružaju.

Evidencija računa se izvodi tako da se postavi jednostavna tablica koja služi kao lista. Ovakav princip je dobar zato što se pažnja korisnika usmjerava na nužne stavke sustava radi jednostavnosti njegove izvedbe.

Proces Računi sadrži četiri procedure i sve su usko vezane s manipuliranjem financija tako da vrše kreiranje računa, brisanje, izmjenu i evidentiranje računa. Potrebne su kako bi se pružila dobra kontrola i preglednost administratoru nad računima i da bi njima jednostavno, sigurno i fluidno pristupio. Kreiranje je izvedeno preko Django funkcionalnosti da formira više računa ili u terminima informacijskih sustava, kreiraju se preko nanizanih unosnih maski pomoću kojih se može kreirati više računa odjednom. Brisanje i izmjena su izvedene preko dugmeta koji prosljeđuju administratora na unosnu masku u slučaju izmjene a u slučaju brisanja na upit administratora za brisanje ili otkazivanje brisanja koje ga vraća na prethodnu stranicu. Evidentiranje računa se vrši preko već spomenutih tablica koje u nizu prikazuju račune.

Funkcija Stanar sadrži dva procesa. Oni su Neplaćeni računi i Profil. Profil je izveden preko predloška koji sadrži dvije funkcionalnosti koje u sustavu raščlanjivanjem objašnjavamo kao procedure a one su unos osobnih podataka i dodavanje profilne slike. Profilna slika se dodaje preko Django ekstenzije *Pillow* koja služi za dodavanje formata slika i se uvoze u konfigurirani statičnu datoteku koja sadržava slike. Kad se kreira profil korisnik

dobiva uobičajenu sliku koju može mijenjati. Unos osobnih podataka izveden je unosnom maskom koja sadrži osnovne podatke stanara kreiranoga tijekom registriranja u sustav i uz te podatke sadrži ostale koje stanar može popuniti i završiti svoj profil. Ovakav način unosa slike i kreiranja profila je uobičajen na skoro svakoj aplikaciji današnjice i zato je dobar način da se i ova aplikacija tako izvede.

Proces Neplaćeni računi sadrži samo proceduru plaćanja. Stanar plaća račune preko dugmeta koji ga prosljeđuje na jednostavni predložak. Na tom predlošku stanar ima dvije opcije. Te dvije opcije su također izvedene preko dva dugmeta koji su povezani s PayPal stranicom. Sustav plaćanja će se podrobnije spominjati u nekim idućim procesima bitnim za rad sustava.

2.3. Radni dijagram

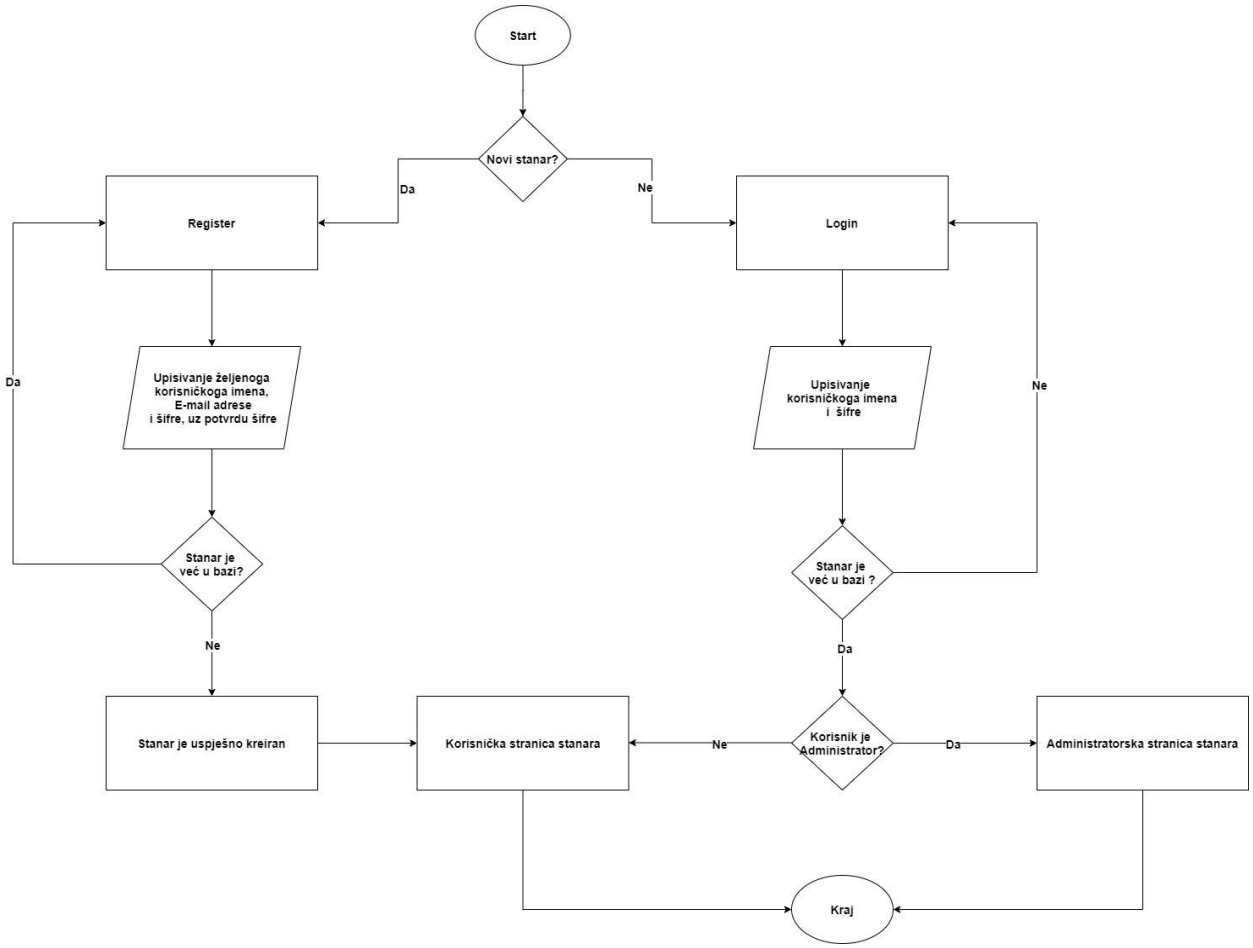
2.3.1. Uvod u radne dijagrame

Radni dijagram je vizualni graf koji pomaže u razlaganju sljedova unutrašnjih struktura tako da prikazuje aktivnosti koje se odvijaju unutar sustava. Mogu biti jednostavni, koji prikazuju jedan segment sustava prikazivan kroz korake s jednostavnim odlukama koje ne vode apstrakciji ili složeni koji prikazuju cijele kompleksnosti uz korake koji svojom odlukom mogu odvesti u neki drugi dio sustava koji sa sobom nosi svoje aktivnosti i u čijim odlukama se mogu definirati neki apstraktniji čimbenici koji bi u budućnosti rada sustava mogli dovesti do određenih pitanja i problema u funkcioniranju sustava.

Bitno je za spomenuti da se neki dijelovi dijagrama ne trebaju nužno nalaziti tj. ne trebaju biti implementirani u informatizirani dio sustava ali mogu utjecati na sustav. Takvi faktori će zasigurno biti uključeni u kompleksnijem dijagramu, kao neki vanjski čimbenici koji imaju sinergiju sa sustavom ali i u jednostavnijem trebalo bi navesti one nužne vanjske čimbenike bez kojih sustav ne bih mogao funkcionirati.

Na sljedećim primjerima će se prikazati osnovni princip registriranja i logiranja stanara ili administratora te dio koji se odvija u sustavu nakon toga. Taj drugi dio predstavlja plaćanje računa korisnika i kreiranja računa od strane administratora. Treba uzeti u obzir da se plaćanje računa može proširiti u općoj primjeni tako da su ostale kompanije sinkronizirane sa sustavom u stilu da umjesto identifikacijskog broja računi daju digitalni pečat računa. Za sada radi jednostavnosti sustava će se prikazati sama srž potrebna za funkcionalnost koja je kreirana unutar aplikacije.

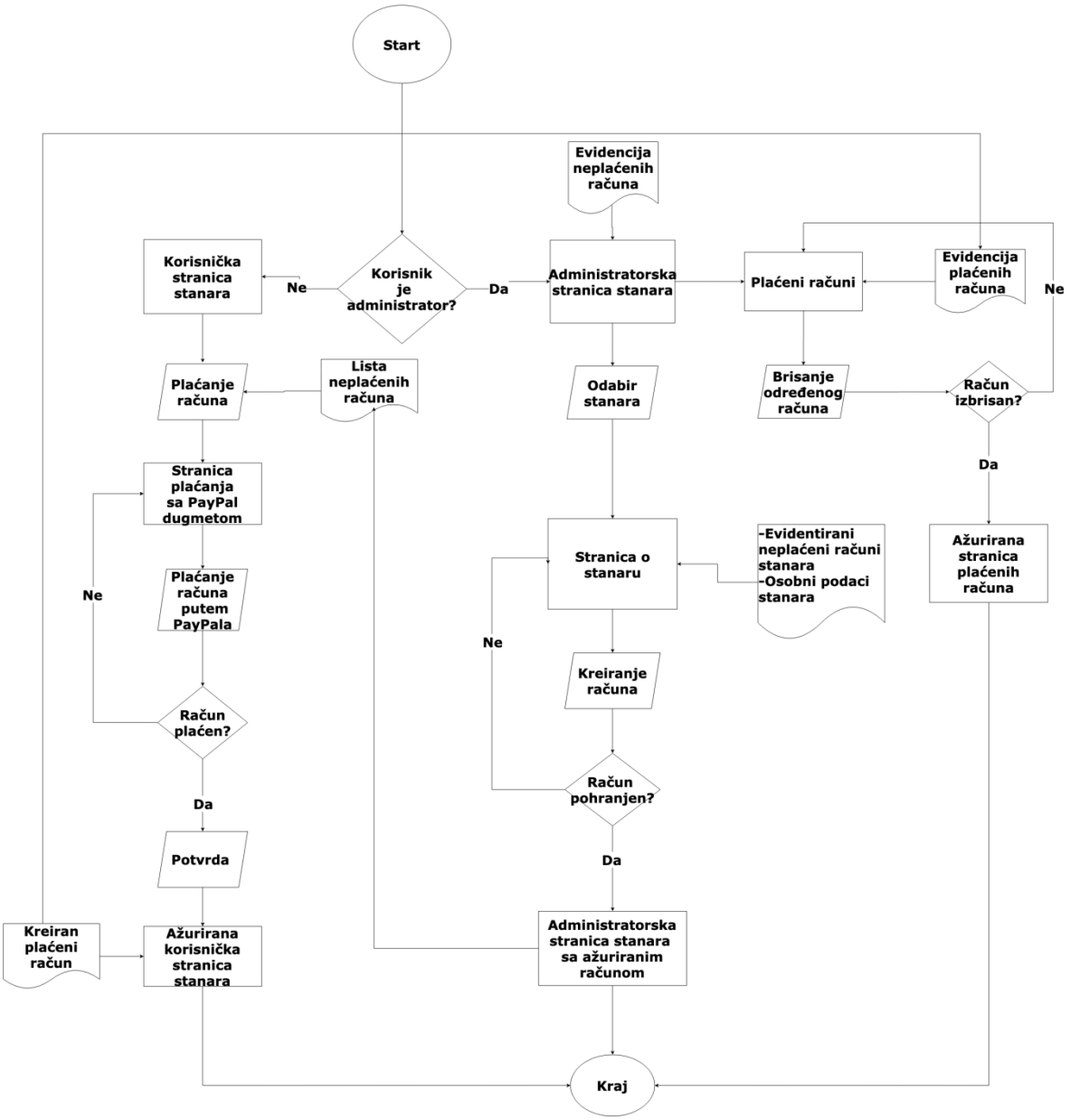
2.3.2. Radni dijagram za spajanje i registraciju korisnika



Slika 2. Radni dijagram za spajanje i registraciju korisnika

Na Slici 2. je prikazan radni dijagram procesa registriranja novoga stanara te spajanja na stranicu postojećega. Ako se korisnik želi registrirati i ući u sustav odabire stranicu za registriranje u čijoj će formi upisati željeno korisničko ime, e-mail te šifru s potvrdom. Ako je taj stanar već u bazi podataka ili već postoji stanar s tim korisničkim imenom unutar sustava, unos se poništava te stanar mora ponovo ispravno ispuniti formu. Nakon uspješnoga registriranja stanar se automatski prosljeđuje na vlastitu dinamičnu stranicu s unikatnim identifikacijskim brojem stanara koju aplikacija stvara nakon uspješnoga registriranja.

2.3.3. Radni dijagram procesa plaćanja i kreiranja računa



Slika 3. Radni dijagram procesa kreiranja i plaćanja računa

Na Slici 3. je prikazan proces plaćanja implementiran unutar aplikacije. Ovaj proces je nastavak prethodnog i započinje od dijela gdje su stanar i administrator prijavljeni u sustavu.

Administrator na svojoj stranici ima uvid u neplaćene račune i popis svih stanara koji su unutar sustava. Uz popis stanara može preko gumba ući u profil stanara. Unutar profila ima mogućnost kreiranja računa te može mijenjati podatke o već unijetim računima i podatke o stanaru. Kad kreira račun automatski je proslijeđen na glavnu stranicu na kojoj se sada nalazi unijeti račun ili više njih a i ti se računi prosljeđuju stanaru kojem su oslovljeni na početnoj stranici stanara preko liste neplaćenih računa. Administrator također može ući i u predložak koji je prikazan na alatnoj traci koji mu daje uvid u plaćene račune i tu ih može brisati preko gumba. Kad izbriše račun sustav ga preusmjerava na taj isti predložak uz jedan račun manje te se te dvije akcije mogu usmjeriti prema kraju radnog dijagrama.

Registrirani stanar na svojoj stranici vidi listu kreiranih računa od strane administratora s atributima od kojih je najbitniji cijena i datum kada je administrator kreirao račun te gumb za plaćanje koji stanara preusmjerava na stranicu plaćanja. Kada stanar klikne gumb, otvori mu se stranica za plaćanje s mogućnosti da plati račun preko PayPala ili kartično. Kada klikne jednu od tih mogućnosti otvori mu se prozor u koji mora ispuniti za uspješno plaćanje. U slučaju PayPala za testiranje napravljeni su sandbox profili za korisnika tj. stanara i primatelja i kad se unesu podatci aplikacija se prosljeđuje na sandbox PayPal. Nakon uspješne transakcije stanaru se umanji svota za postavljenu dinamičnu vrijednost a na drugom poslovnom računu administratora se uveća za istu. Korisnik dobije alarmnu poruku o uspješnosti transakcije te klikom na poruku se prosljeđuje natrag na svoju početnu stranicu na kojoj je sada izbrisan račun s neplaćenim statusom, te je kreiran ispis s datumom koji je automatski proslijeđen administratoru na njegovu stranicu s plaćenim računima.

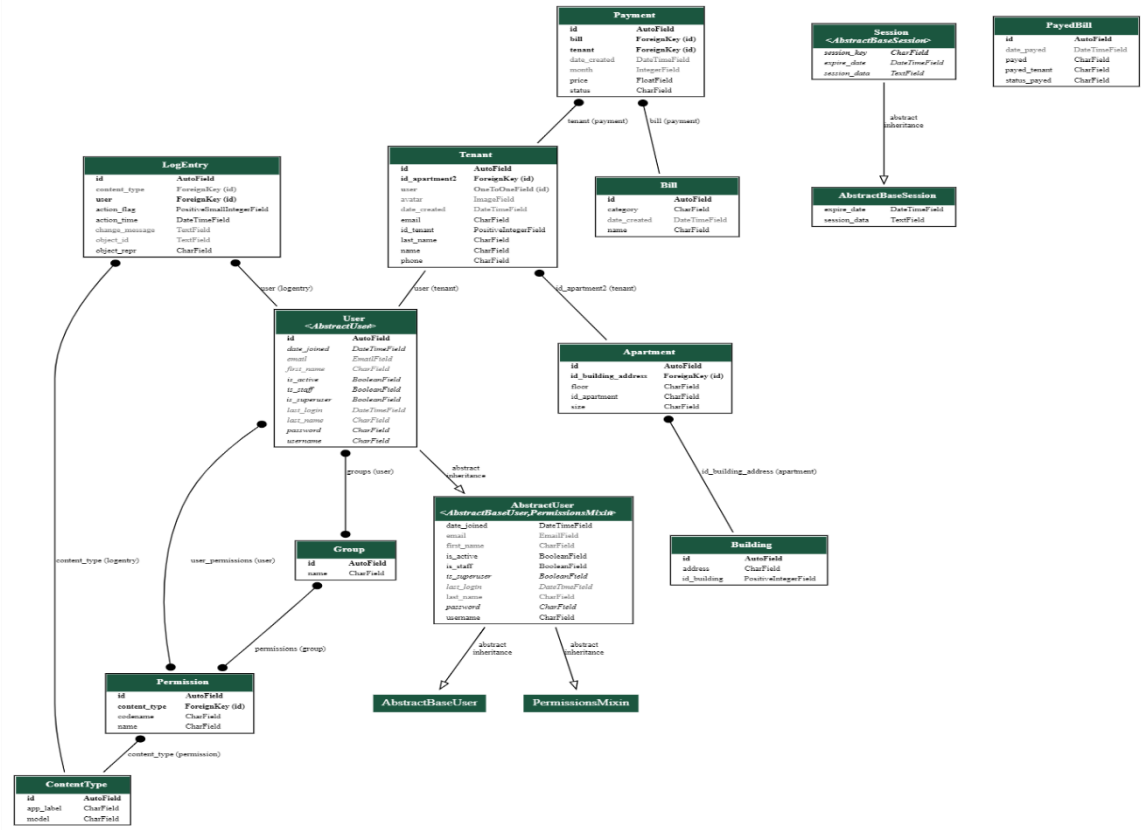
Zaključak je da se dvosmjernim putem rješava problem plaćanja, tako da računi koje kreira administrator imaju status da nisu plaćeni te se prosljeđuju stanaru a stanar plaćanjem briše taj isti račun te se kreira novi „račun“ koji služi administratoru kao uvid da je račun plaćen. S tim je ova aktivnost dovršena te se prosljeđuje na kraj.

2.4. Model podataka

2.4.1. Kreiranje modela podataka Djangom

Django ima funkcionalnost da se uz dodanu ekstenziju kreira model podataka u *dot* formatu ili *png* formatu kao vizualni prikaz i to se vrši preko komandne linije. Kreirana slika pokazuje sve dodane modele unutar datoteke *models.py* odnosno klase kako se općenito nazivaju a tako i u terminu informacijskih sustava.

Model podataka se kreira tako da se komandnoj liniji treba upisati `pip install django-extensions` i treba ih dodati u *settings.py* unutar instaliranih aplikacija `INSTALLED_APPS`. Nakon toga se unosi komanda `python manage.py graph_models -a > output.png`. Ako se ne može kreirati *png* potrebno je instalirati *graphviz* a ako ni to ne funkcionira, treba dodati *graphviz* na PATH unutar Windowsa. Slika prikazuje kreirani model podataka.



Slika 4. Relacijski model podataka aplikacije

2.4.2. Zadane klase u Djangu

Svaka tablica u bazi podataka je u aplikaciji predstavljena klasom, čija svojstva predstavljaju atribute. Atributi prezentiraju tip podatka jednoga entiteta i vrijednost tog entiteta koju unosimo u bazu podataka.

Centralna klasa ovog sustava je klasa *User* koja je u aplikaciji uvezena iz Django biblioteke te je korisnik ne kreira već prosljeđuje na željeni model odnosno klasu. Ona sadrži polja odnosno entitete, atribute i metode koje imaju praktične funkcionalnosti za autentifikaciju, dozvole i ostale metode koje se koriste tijekom daljnjeg razvoja aplikacije. Klasa Tennant nasljeđuje klasu User, što znači da, uz vlastite atribute, nasljeđuje sve atribute klase User i ostala njena svojstva.

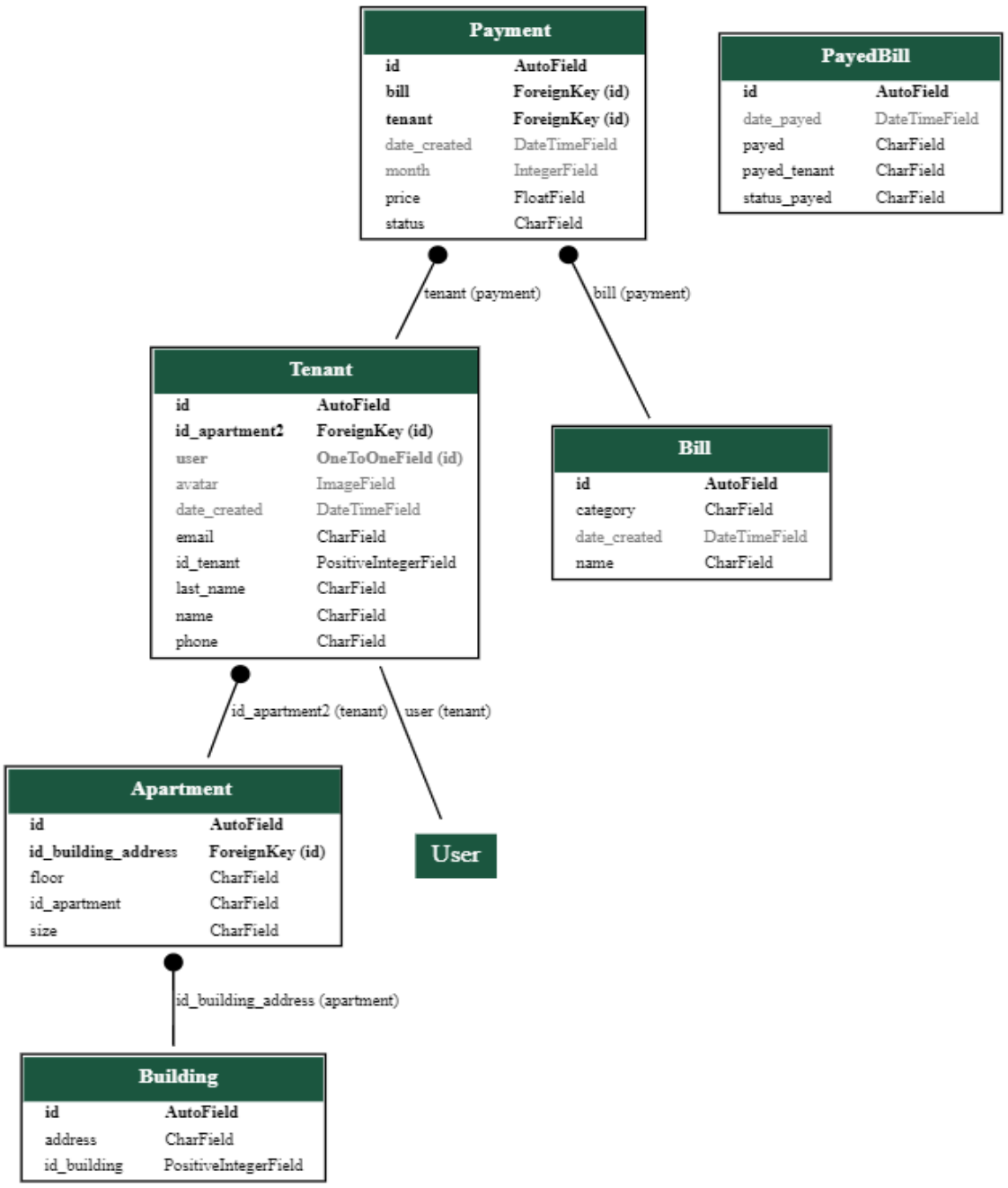
Klasa Abstract User služi kao klasa koja sadrži svojstva autentifikacije kako glavna klasa, User, ne bi bila pretrpana funkcionalnostima i svojstvima unaprijed određenima unutar Django biblioteke. Većinu spomenutih funkcionalnosti klasa User nasljeđuje od klase Abstract User.

Klasa Group služi za grupiranje korisnika kreiranima unutar klase User ako administrator želi grupirati korisnike. Povezana je s klasom User s relacijom više naprema više što znači da se korisnici mogu grupirati u više željenih grupa.

Klasa Log Entry služi kao dnevnik koji prati sve promjene default modela i povezana je s klasom User relacijom više naprema jedan.

Klasa Content Type pohranjuje informacije o kreiranim modelima. Kada korisnik napravi novi model odnosno klasu i implementira je preko komande, ta klasa se automatski pohranjuje unutar klase Content Type. Ona je relacijom jedan naprema više spojena s klasom Log Entry i s klasom Permission.

Klasa Permission služi za određivanje prava korisnicima ili grupama stoga su relacijski povezani više naprema više s klasama User i Group.



Slika 5. Klase kreirane u okruženju Django.

2.4.3. Klase kreirane automatski u Django okruženju

Korisnik generira vlastite klase korištenjem django alata manage, na sljedeći način: potrebno je upisati kod unutar komandne linije `python manage.py graph_models app_building –o mymodels.png`.

Podebljani atributi s oznakom *(id)* na Slici 4. i na Slici 5. predstavljaju identifikatore i koriste se za ostvarivanje relacijske veze s ostalim klasama.

Klasa Tenant predstavlja stanara i kao što je spomenuto, ona nasljeđuje sva svojstva Django default klase User da bi se moglo lakše manipulirati s podacima vezanim za stanare unutar Django admin stranice. Povezana je relacijom jedan naprema više s klasom Payment u kojoj administrator kreira račune. To znači da jedan stanar može imati više računa a jedan unikatan račun ne može biti proslijeđen prema više stanara.

Klasa Bill predstavlja vrste troškova s entitetima, id koja je identifikacijski indeks troška , opis troška koji opisuje da li trošak spada u trošak zgrade ili trošak stana, datum kada je trošak kreiran te ime troška. Klasa je povezana relacijom jedan naprema više s klasom Payment. To znači da jedna vrsta troška može biti generirana na više računa te s child-parent upitom možemo izvući bitan atribut kao opis troška koji je tako moguće prikazati na ekranu da se zna o kojem se opisu radi.

Klasa Payment služi kao račun stanaru koje kreira administrator. Sadrži entitete poput datuma kreiranja računa, mjeseca za koji se plaća račun te je ograničen kao integer da bude prikazan kao padajuća lista raspona od 1 do 12. Sadrži još cijenu koju administrator upisuje te status koji je određen kao default neplaćen da bi se preko mogućnosti filtriranja unutar aplikacije mogao prepoznati neplaćeni račun te pomoću brojača zbrajati i tim prikazati uz izvlačenje određenog atributa ili svih entiteta dug određenoga stanara ili općeniti dug. Entiteti bill i tenant unutar konfiguracije svojih atributa sadrže relacijsku povezanost sa spomenutim klasama Bill i Tenant.

Klasa Building predstavlja zgradu koja sadrži samo entitete id zgrade i adresu. Relacijski je povezana s klasom Apartment s relacijom jedan naprema više što znači da jedna zgrada ima više stanova koji ne mogu pripadati svojim id-om niti jednoj drugoj zgradi.

Klasa Apartment predstavlja stan s entitetima id stana, kat, veličina stana te na kraju id zgrade s kojim je relacijski povezan. A povezan je još sa stanarom s relacijom jedan

naprema više što znači da više stanara može stanovati u jednom stanu i ne mogu biti dio nekoga drugoga. Pošto je stanar povezan s računima može se također child-parent upitom dobiti dugovanje stana.

3. Razvojno okruženje Django

3.1 Općenito o Djangu

Django je jedan od vodećih mrežnih razvojnih okruženja (engl. *web framework*) na svijetu. Vrlo je popularan zbog praktičnosti i jednostavnosti upotrebe. Besplatan je i otvorenoga koda s toga svako ga može koristiti, proučavati i izmjenjivati. Dizajniran je da pomogne korisnicima da razviju aplikaciju brzo od ideje do realizacije.

Django je razvijen između 2003. i 2005. godine od strane web programera Adriana Holovaty-a i Simona Willisona koji su stvorili i održavali novinarsku internetsku stranicu *Lawrence Journal-World*. Nakon brojnih stranica tim je prepoznao učestali kod i kreirao uzorak. Taj kod je evoluirao u općeniti mrežni okvir koji je bio predstavljen kao „*Django*“ projekt u srpnju 2005.

Django korisniku omogućava lako održavanje, jednostavan i čist kod i brže debugiranje. Otklanja kompliciranost kodiranja što ga čini veoma popularnim među nekima od vodećih aplikacija u svijetu. Koriste ga neke popularne novinarske web stranice poput *New York Times*-a, *The Guardian* i *The Washington Post*. Također ga koriste aplikacije kao što su *Mozilla*, *Instagram*, *Spotify* i mnoge druge.

3.2 Instalacija

Izrada aplikacije započinje instaliranjem programskog jezika Python u prozoru naredbenog retka tj. komandne linije (eng. *command prompt*) ili u *PowerShell*-u ako se koristi *Windows*. *Linux* alternative su *Gnome, Guake, Xterm, Konsole, Terminator* i mnoge druge. Ako je *Python* instaliran pošto se uobičajeno već nalazi na software-u prelazi se na sljedeći korak.

Poželjno je da prije instalacije korisnik kreira direktorij koji sadrži Pythonovo virtualno okruženje (eng. *virtual environment*) i da se unutar te mape kreira daljnji projekt. Virtualno okruženje se instalira tako da u komandnoj liniji zadamo komandu `pip install virtualenv` i nakon toga ako korisnik želi kreirati mapu sa skriptama, bibliotekama i ostalim podacima virtualnoga okruženja, korisnik treba upisati komandu `virtualenv imeokruženja`. Nakon kreiranja mape slijedi aktivacija koja se nalazi u mapi *Scripts*. Unutar te mape u komandnoj liniji upisuje se `.\activate` i tako korisnikov rad računala i software ostaje zaštićen.

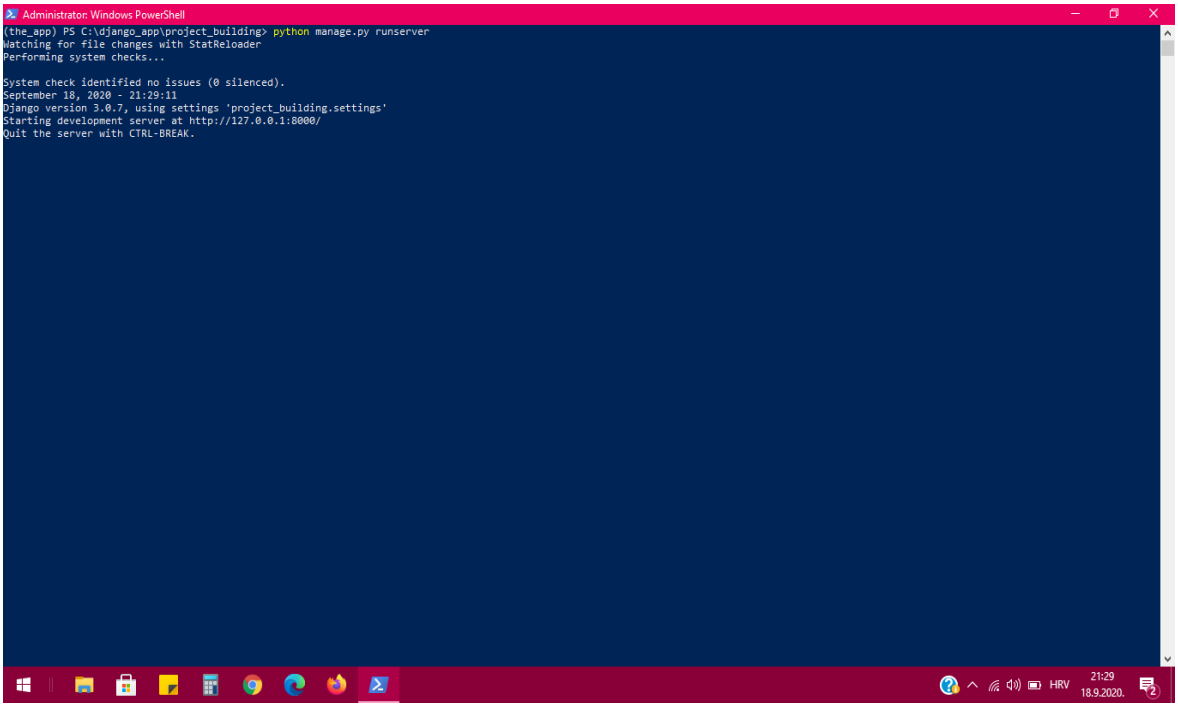
Instalacija Djanga kreće tako da u željenom direktoriju pomoću komande `pip install django` pokrećemo instalaciju. Nakon instalacije pomoću komande `django-admin startproject imeprojekta` kreira se novi Django projekt. Poslije kreacije projekta potrebno je preko komandne linije ući u kreiranu mapu koja je u ovom slučaju nazvana imeprojekta. Projekt koji je kreiran u sebi sadrži Python podatke za postavke, putanje i ostale Django podatke potrebne za osnovni rad projekta.

Najbitniji segment unutar projekta je *manage.py* pomoću kojega pokrećemo komande za daljnji razvoj aplikacije i on mora ostati nepromjenjiv. Unutar mape imeprojekta u naredbenom retku aplikacija se kreira upisivanjem komande `python manage.py startapp imeaplikacije`. S tim je kreirana aplikacija `imeaplikacije` koja je spremna za buduće postavke.

3.3. Kreiranje administratora

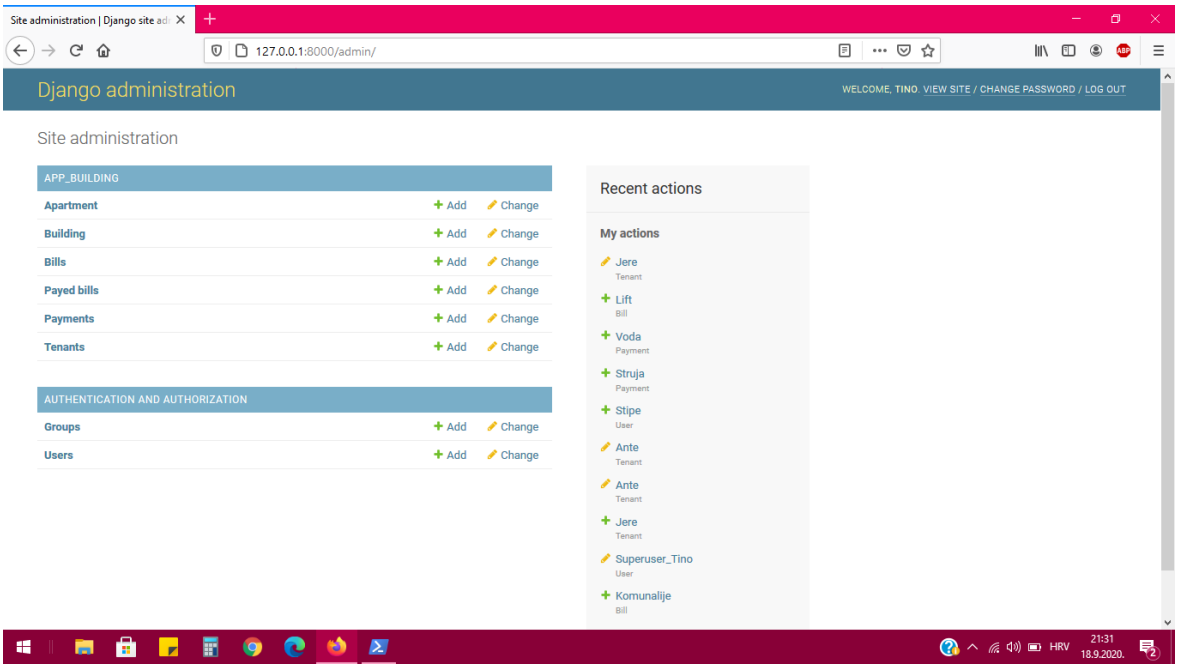
Django sadrži dobru i pouzdanu administratorsku stranicu koja je već implementirana samim instaliranjem. Takva stranica je snažan alat pogotovo u početku razvijanja aplikacije zbog toga što je vizualno povezana sa srži aplikacije tj. modelima od kojih prvo kreće razvoj na ovakvom frameworku i ostalima sa sličnom arhitekturom. Vizualno model u Djangu označava klasu koju možemo definirati kao naslov tablice. Ta tablica sadržava varijable određene kao tekst, broj ili neka druga definirana vrijednost i metode koje služe kao veze između dvije ili više tablica po pravilu koje zadajemo.

Korisnik koji će imati pristup već zadanoj (eng. *default*) administratorskoj stranici se kreira upisivanjem komande `python manage.py createsuperuser` . Nakon toga se upisuje željeno ime, E-mail adresa korisnika i šifra te s tim je kreacija korisnika sa svim pravima dovršena. Unosom kôda `python manage.py runserver` pokreće se server i na pregledniku se upisuje lokalna adresa (eng. *localhost*) uz slobodni port koji je obično 8000. Na taj link se upisuje `/admin` koji je unaprijed određen u kreiranome projektu pod *Python* datotekom *urls.py* . Korisnik unosi podatke koje su kreirani preko komande i s tim je proces dovršen.



Slika 6. Pokretanje Django servera iz komandne linije

Sada korisnik ima vizualnu predodžbu modela ako su uneseni i svih veza s kojima su povezani. Preko već zadanoga modela *Group* može kreirati neku novu s manje administratorskih prava i kasnije vezati s nekima od stvorenih modela da bi ograničio prava na jednostavan način. Drugi default model je *Users* u kojem je korisnik već uključen kreiranjem profila preko komande `createsuperuser`. Unutar toga modela mogu se kreirati novi korisnici kojima možemo dodjeljivati prava. Na kraju je nužno za spomenuti da korisnik ne bih trebao koristiti ovaj alat za sveukupno razvijanje i oko njega graditi vanjsko sučelje (eng. *frontend*), već kao vizualno pomagalo za budući razvoj aplikacije ali u jednome trenutku za kompletnu aplikaciju bi korisnik trebao kreirati sam kreirati svoju administratorsku stranicu.



Slika 7. Django administratorska stranica

Slika 7. prikazuje Django administratorsko sučelje. Prvi dio s lijeve strane prikazuje klase, odnosno Django modele kreirane od strane korisnika a drugi dio prikazuje Django uobičajene modele koji korisniku služe za autentifikaciju u početnom razvoju aplikacije.

4. Osnovna konfiguracija

4.1. Url

4.1.1. Osnovne postavke

Konfiguracija kreirane aplikacije se ostvaruje preko urednika (eng. *editor*) kojeg korisnik upotrebljava. Primjeri urednika su *Visual Studio*, *Sublime Text*, *Atom*, *Notepad++* i mnogi drugi. Za ovu određenu aplikaciju koristimo Sublime Text.

Prvo je potrebno nakon otvaranja urednika otvoriti mapu koja je kreirana u komandnoj liniji. Mapa projekta sadrži postavke settings.py i unutar se dodaje kreirana aplikacija tako da se upisuje `'imeaplikacije',` ispod kôda `INSTALLED_APPS` gdje se definiraju sve aplikacije koje se instaliraju u razvoju projekta. Nakon implementacije unutar postavki projekta potrebno je kreirati datoteku *urls.py* u mapi instalirane aplikacije. Projekt mapa također sadrži urls.py ali ona za sada ostaje nepromijenjena zbog toga što taj dio sadrži putanju za već postavljenoga i podešenoga administratora s unaprijed ugrađenim sučeljem kreiranim instalacijom Djanga.

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'app_building.apps.AppBuildingConfig',
    'django_filters',
    'stripe',
    'django_extensions',
    'graphviz',

]
```

Slika 8. Instalirane aplikacije unutar settings.py

Slika 8. Prikazuje sve uobičajeno instalirane Django aplikacije i one koje korisnik instalira te unosi unutar datoteke.

Nakon kreiranja urls.py na vrhu treba upisati kôd `from django.urls import path` a to je uobičajena Django funkcija koju koristimo za konfiguraciju usklađenog lokatora sadržaja (eng. *Uniform Resource Locator*) poznatiji pod kraticom URL. Nužno je još uvesti *views.py* datoteku koja će se najviše koristiti za obradu i provođenje podataka koju ta datoteka vrši. Datoteku se uvozi kôdom `from . import views`. Razlog zašto se samo upisuje točka je što se ne uvozi datoteka iz baznoga direktorija Djanga već iz baznoga direktorija u kojemu se nalazi instalirana aplikacija. Ispod *importa* se upisuje isti kôd kao i u mapi projekta no u zagradama nam se nalazi putanja koju korisnik definira i prikazat će se na URL-u browsera imenom koje smo naveli u navodnicima. Nakon putanje slijedi drugi argument koji će nam slati podatke iz *views.py* i upisujemo *views.homepage*. Nakon toga potrebno je uključiti (eng. *include*) kreirani *urls.py* iz aplikacijske mape s uobičajenim *urls.py* mape projekta da bi bile povezane.

4.1.2. Dinamični URL

Dinamični url kroz upit vraća url koji je povezan s tim zadanim upitom (eng. *query*). U Djangu takvi upiti se kreiraju u datoteci *views.py*. Na ovaj način radi svaka stranica koja ima svoju bazu podataka. U upitima se izvlače informacije o modelu iz *models.py*. Razlika između spomenute statične stranice i dinamične je u tom što statična ostaje nepromijenjena i ne sadrži unikatne elemente za primjerice korisnika a dinamična je automatizirana i generira se preko upita povezanima s bazom podataka i stoga se mogu kreirati raznorazni profili.

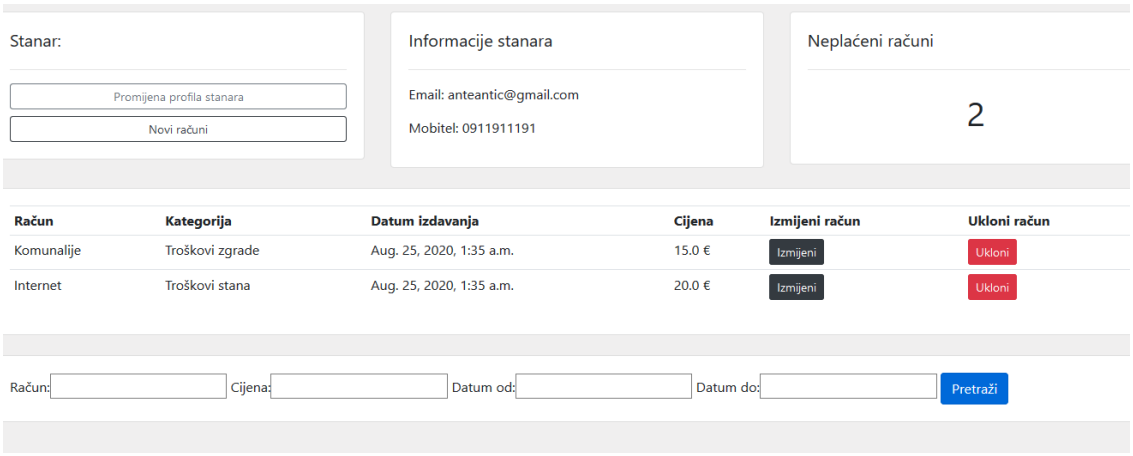
Django stranicu čini dinamičnom tako da je potrebno u *views.py* od definiranoga viewa kojega za primjer zovemo *ime_viewa* uz argument *request* se dodaje još i argument *pk*. Taj *pk* predstavlja primarni ključ. Nužno je navesti da primarni ključ se može nazivati i drugačije ali radi estetike bit će nazvan *pk*. Ispod definirane funkcije s primarnim ključem, taj ključ uključujemo u upit na način `ime_queryja = ImeModela.objects.get(id=pk)`.

Nakon kreiranja primarnoga ključa u views.py mora se podesiti putanja (eng. *path*) u urls.py od aplikacije. To se izvodi tako se upisuje kôd `path('ime_viewa/<str:pk>/', views.ime_viewa)`. Prvi argument u navodnicima koji označava putanju se također ne treba nazivati kao definirani *view* ali dobra je praksa da se putanja i *view* nazivaju kao model, pošto obično vrše funkciju toga modela na stranici koja se putanjom zadaje.

```python
from django.urls import path

from django.contrib.auth import views as auth_views

from . import views

urlpatterns = [
    path('register/', views.registerPage, name="register"),
    path('login/', views.loginPage, name="login"),
    path('logout/', views.logoutUser, name="logout"),

    path('bills/', views.bills, name="bills"),
    path('', views.expenses, name="expenses"),
    path('tenant/<str:pk>', views.tenant, name="tenant"),
    path('update_tenant/<str:pk>', views.updateTenant, name="update_tenant"),

    path('user/', views.userPage, name="user-page"),
    path('user_settings/', views.userSettings, name="user_settings"),


    path('create_bill/<str:pk_two>/', views.createPayment, name="create_bill"),
    path('update_bill/<str:pk_two>/', views.updatePayment, name="update_bill"),
    path('delete_bill/<str:pk_two>/', views.deletePayment, name="delete_bill"),
    path('delete_payed/<str:pk_two>/', views.deletePayed, name="delete_payed"),


    path('card/<str:pk_two>/', views.card, name="card"),
    path('complete/', views.paymentComplete, name="complete"),

]
```

Slika 9. Prikaz putanja unutar urls.py

Slika 9. prikazuje putanje unutar kreirane aplikacije. Dinamične putanje su obično one gdje korisnik mijenja podatke. U ovom slučaju se primjenjuju za kreiranje, mijenjanje i brisanje računa te mijenjanje podacima o stanaru.



Slika 10. Prikaz dinamične stranice stanara

Slika 10. prikazuje dinamičnu stranicu stanara gdje administrator može mijenjati podatke o stanaru i dodavati račune. Administrator u sredini ima prikaz računa od određenog stanara koje može mijenjati ili brisati. Donji dio sadrži implementirani Django filter pomoću kojega se može filtrirati željeni račun.

4.2. Predlošci

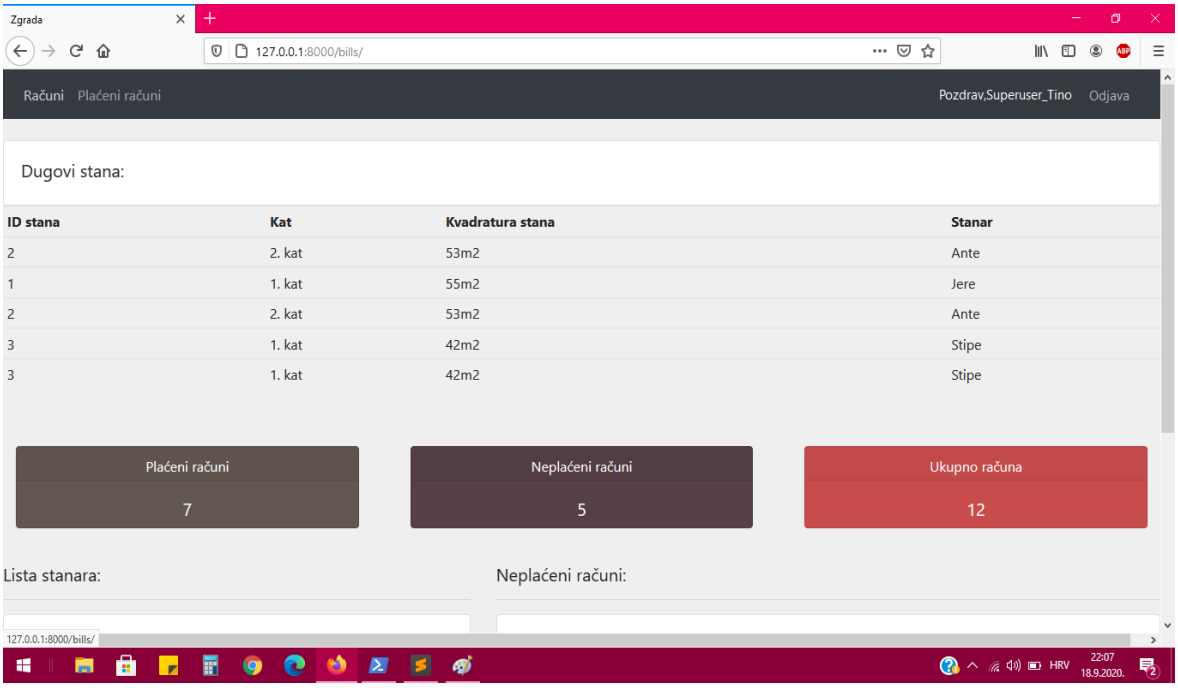
4.2.1. Osnovni predlošci

Predložak (eng. *template*) je datoteka u kojoj se kreira grafičko sučelje pomoću programa poput HTML-a, CSS-a, Bootstrapa i ostalih programa orijentiranih za *frontend* razvijanje aplikacije. Django ima unikatan način za pohranjivanje predložaka u datoteku.

U mapi aplikacije treba napraviti novu datoteku *templates* i tako se treba nazivati pošto će Django točno tražiti putanju toga naziva. Unutar mape *templates* kreira se nova mapa koja nosi naziv kao aplikacija i unutar se pohranjuju datoteke. Dodavanjem nove datoteke unutar mape koja je kreirana za predloške unosimo za sada osnovni HTML kod i spremamo ga po nazivom *homepage.html* . Nakon toga u *views.py* uz zadanu metodu definira se ime upisivanjem kôda `def homepage(request):` te ispod slijedi kôd koji vraća tu kreiranu datoteku `return render(request, 'imeaplikacije/homepage.html').` Taj postupak se ponavlja za svaki predložak koji u *views.py* moramo definirati i proslijediti na *urls.py* gdje se unosi putanja.

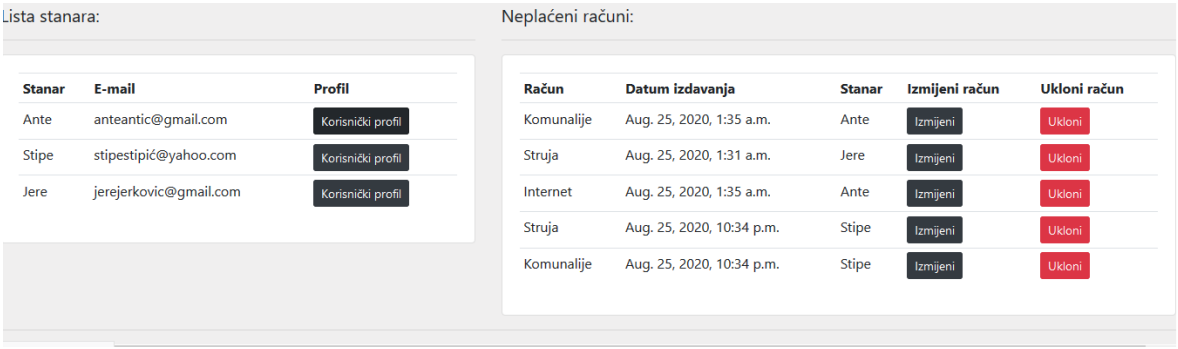
Ako je potrebno da se naslijedi neki segment kôda iz predloška tj. da ga se uključi iz jednog predloška u neki drugi ili sve ostale za to postoji Django metoda. Ta metoda se realizira da se kreira nova *template* datoteka koja će predstavljati glavni kod koji sve ostale HTML datoteke nasljeđuju i obično će se nazivati *main.html*. Ta metoda se ostvaruje da na željenome mjestu unutar kreiranoga predloška se upiše kôd `{% block naziv %}` i zatvara sa `{% endblock %}` na začelju idućega predloška u kojemu je potrebno uključiti sadržaj glavnoga predloška se upisuje `{% extends 'imeaplikacije/main.html'%}` a ispod se stavlja spomenuti kôd block i endblock između kojega upisujemo kôd stranice što znači u ovom slučaju da će se sadržaj stranice main ispisati u gornjemu dijelu stranice koja sadržaj nasljeđuje a ispod njen sadržaj.

Za kreiranje navigacijske trake (eng. *navigation bar*) većinom se koristi druga metoda i stoga slijedi kreiranje predloška naziva *navigation.html* koji se koristi u primjeru. Ako je potrebno uključiti cijeli dio kôda bez ograničavanja koristi se metoda kôda `{% include 'imeaplikacije/navigation.html' %}`. Metoda *extends* stvara *parent child* relaciju i postoji mogućnost za promjenu informacije dok *include* procesira html odgovor.



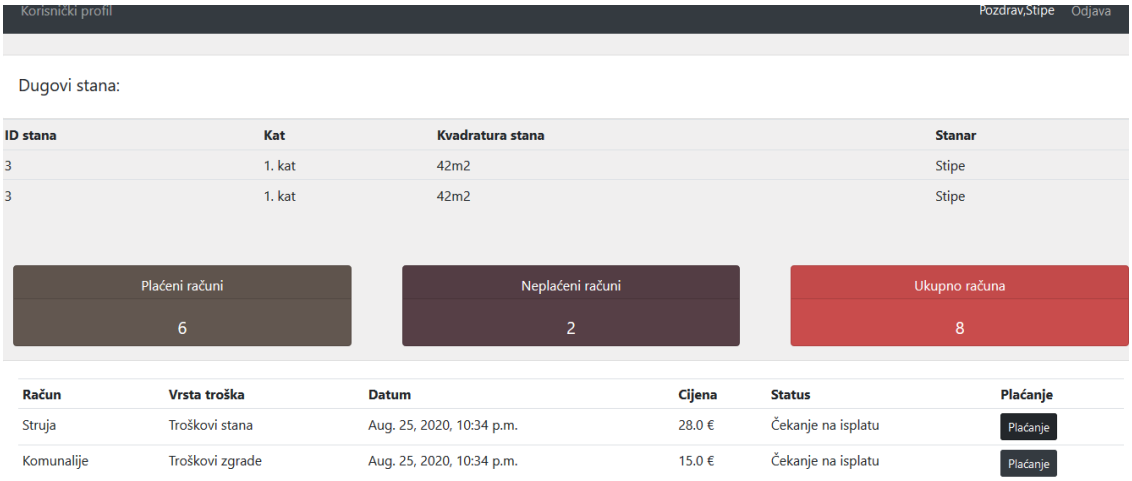
Slika 11. Glavna stranica administratora 1. dio

Na Slici 11. Prikazana je glavna stranica administratora koja sadrži listu računa po broju stana da administrator može pratiti dug stana i tablicu koja je prikazana karticama koje prikazuju koliko trenutno ima plaćenih, neplaćenih i ukupan broj računa. Donji dio sadrži listu stanara gdje administrator može ulaziti u profil stanara i neplaćene račune koje može mijenjati. Gornji dio se nalazi u zasebnom predlošku koji je uključen u drugi predložak koji sadrži listu stanara i neplaćene račune. To je postignuto spomenutom metodom *include*. Navigacijska traka na vrhu sadrži dvije stranice i gumb u desnom gornjem kutu za odjavu prijavljenoga korisnika.



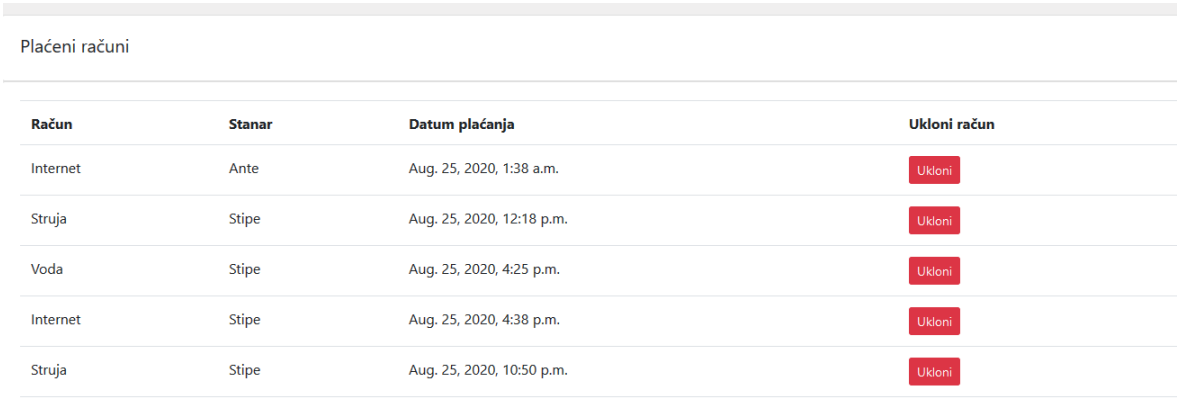
Slika 12. Glavna stranica administratora 2. dio

Slika 12. sadrži donji dio glavne stranice administratora. Sastoji se od dvije tablice. U prvoj tablici administrator vidi listu stanara i osnovne podatke o stanaru. Treći redak sadrži gumb koji administratora prosljeđuje na korisnički profil stanara prikazan na slici 10. Druga tablica sadrži sve neplaćene račune od stanara koje administrator može izmjenjivati ili brisati ako je nešto krivo uneseno pri kreiranju računa.



Slika 13. Glavna stranica stanara

Slika 13. prikazuje glavnu stranicu od stanara. Sadrži iste tablice kao stranica od administratora ali pomoću dinamičnosti koja svakom stanaru daje unikatni identifikacijski broj ne sadrži sve podatke, nego samo one koji su vezani za prijavljenoga stanara. Stanar ne može mijenjati račun već ga samo plaćati i kad stanar klikne gumb za plaćanje, on se prosljeđuje na drugu stranicu čiji je postupak objašnjen u prethodnom poglavlju. Navigacijska traka stanara sadrži korisnički profil gdje stanar može mijenjati svoje podatke.



Slika 14. Stranica plaćenih računa

Slika 14. Prikazuje listu plaćenih računa. Svaki plaćeni račun se kreira unutar ove stranice nakon što stanar plati račun i onda administrator na listi ima uvid kad je račun plaćen i tko ga je platio. Nakon uspješnog plaćanja, račun sa slike 12. i slike 13. se briše i kreira se plaćeni račun s datumom plaćanja što daje uvid administratoru u obliku ove liste.

4.2.2. Oznake predloška

Oznake predloška(eng. *template tags*) u Djangu služe kao alat za dohvaćanje podataka iz baze preko upita zadanoga u datoteci *views.py*. Tako se definira što će se prikazivati unutar predloška i na koji način će se taj podatak obraditi. To znači da se uzima zadana vrijednost iz *views.py* određena upitom a taj dio u *views.py* sadrži upit vezan za neki od modela iz datoteke *models.py.*

Koriste se najčešće dvije vrste takvih oznaka koje su općenito poznate u programerskom svijetu. *If* oznaku koja provjerava da li je određeni zadani dio iz modela *True* ili *False,* i *For* oznaku kao petlju. Treba zapamtiti da prije svega toga te datoteke trebaju biti povezane preko importa na začelju svake.

Za *For* oznaku treba unutar predloška na željeno mjesto upisati kôd `{% for ime_oznake in ime_viewa %}` ispod se piše kôd koji je uvučen tabulatorom `{{ime_oznake.ime_atributa}}`. U primjeru `ime_atributa` označava željeni atribut iz modela koji je određen upitom u views.py i sve se zatvara sa `{% endfor %}`.

If oznaka se piše kôdom `{% if ime_atributa %}` a zatvara se sa `{% endif %}`. Mogu se kombinirati ove dvije metode što je dobra stvar za dinamičnost stranice. Unutar možemo i koristiti *and*,*or* i *not*. `{% elif %}` se koristi ako postoji više argumenata za True vrijednost i `{% else %}` predstavlja False vrijednost. If sadrži i operatore `==` za jednakost, `!=` za različitost, `<` za manje od, `>` za više od, `<=` za manje ili jednako te za više ili jednako operator `>=`.

Podaci se prosljeđuju u *views.py* s kodom ispod definiranog *view*-a te se piše `ime_viewa = ImeModela.objects.all()`. Ovo znači da se uzimaju svi atributi iz modela ImeModela. Ispod treba napisati kôd što funkcija vraća `return render(request, 'ime_aplikacije/ime_predloška.html', {'ime_viewa': ime_viewa}`.

```
<br>

<div class="row">
    <div class="col-md">
        <div class="card card-body">
            <h5>Dugovi stana:</h5>
        </div>
            <table class="table table-sm">
                <tr>

                    <th>ID stana</th>
                    <th>Kat</th>
                    <th>Kvadratura stana</th>
                    <th>Stanar</th>



                </tr>
                {% csrf_token %}
                {% for payment in pay %}

                <tr>
                        <td>{{payment.tenant.id_apartment2}}</td>
                        <td>{{payment.tenant.id_apartment2.floor}}. kat</td>
                        <td>{{payment.tenant.id_apartment2.size}}
                        <td>{{payment.tenant.name}}</td>




                </tr>

                {% endfor %}


            </table>
        </div>
    </div>
</div>

<br>
<br>
```

Slika 15. Predložak glavne stranice administratora

4.3. Modeli

4.3.1. Osnovni modeli

Django pohranjuje modele u datoteci *models.py* i po definiciji te datoteke su klase koje nasljeđuju podatke *models* iz Django baze. Najčešće se za atribute koristi *integer* ako je potreban rad s cijelim brojevima, za tekst koristimo *string* i za decimalne brojeve koristimo *float* ako imamo nekakve cijene. Modele koristimo osnovnu bazu podataka željene aplikacije.

Osnovni model kreira se upisivanjem kôda `class Ime_modela(models.Model)`. Unutar zagrade se upisuje što se nasljeđuje i od kuda. Ispod se unose atributi i upisuje se `ime_atributa = models.CharField(max_length=100)`. Maksimalna duljina stringa se mora odrediti po defaultu. Ako je riječ o brojci upisuje se `IntegerField` a ako je riječ o decimalnoj brojci upisuje se `FloatField` . Za dodavanje trenutnoga datuma koristi se `DateTimeField` i da bismo dobili automatsko vrijeme kad se nešto kreira unutar klase u kojoj se ovaj atribut nalazi u zagradi se upisuje `auto_now_add` i postavljamo na vrijednost `True`. Za polja koja mogu ostati prazna dodajemo kôd `null = True`. Ako je potrebno da model na administratorskoj stranici vraća jedan od atributa, ispod modela se upisuje kôd `def__str__(self):` za string i ispod uvučeno tabulatorom upisuje se `return self.ime_atributa` za vraćanje u ovom slučaju imena atributa.

Da bi se klasa spremila u bazu podataka prvo se treba u naredbenom retku upisati komanda `python manage.py makemigrations`. Ova komanda priprema bazu podataka za migraciju tako da kreira promjenu u mapi *migrations*. Nakon pripreme se upisuje komanda `python manage.py migrate` koja pokreće SQL kod iniciran spremanjem migracije unutar mape. Sada ponovnim logiranjem na admin stranici još se ne može uočiti promjena jer je potrebno u datoteci admin.py registrirati promjenu. Prvo se uvozi kreirani model iz modela da bi se spojile dvije datoteke a ispod se upisuje kôd `admin.site.register(Ime_modela)`. Sada pokretanjem servera i ulaskom u stranicu administratora uočavamo kreirani model.

4.3.2. Relacijski modeli

Kreiranje relacijskih modela u Djangu se jednostavna i to je još jedan dobar primjer njegove praktičnosti. Django podržava relacijsku bazu podataka i dopušta da se modeli povezuju na tri načina, a oni su: *One-to-one*, *Many-to-many* i *One-to-many*.

One-to-one se koristi kada se jednom modelu dodjeljuje točno samo drugi model. Ovakav tip može biti koristan kao primarni ključ drugoga objekta ako mu proširuje vrijednosti svojim atributima na nekakav način. U Djangu je definiran kôdom `OneToOneField`.

Many-to-many se koristi kada jedan model može nasljeđivati svojstva od više modela a to vrijedi i obrnuto tj. taj model od kojega nasljeđuje svojstva također nasljeđuje atribute ostalih modela s kojima je povezan. U Djangu je definiran kôdom `ManyToManyField`.

One-to-many relacija se koristi kada se jedan model relacijski povezuje s više njih. Takva veza je definirana kôdom `ForeignKey`.

```
class Tenant(models.Model):
    user = models.OneToOneField(User, null=True, blank=True, on_delete=models.CASCADE)
    id_tenant = models.PositiveIntegerField(default=1)
    id_apartment2 = models.ForeignKey(Apartment, null=True, verbose_name="Number of the apartment", on_delete=models.SET_NULL)
    name = models.CharField(max_length=100, null=True)
    last_name = models.CharField(max_length=100, null=True)
    email = models.CharField(max_length=100, null=True)
    phone = models.CharField(max_length=100, null=True)
    avatar = models.ImageField(default="avatar.png", null=True, blank=True)
    date_created = models.DateTimeField(auto_now_add=True)


    def __str__(self):
        return self.name


class Bill(models.Model):
    CATEGORY = (
        ('Troškovi zgrade', 'Troškovi zgrade'),
        ('Troškovi stana', 'Troškovi stana'),
        )

    name = models.CharField(max_length=100, null=True)
    date_created = models.DateTimeField(auto_now_add=True)
    category = models.CharField(max_length=100, null=True, choices=CATEGORY)

    def __str__(self):
        return self.name

class Payment(models.Model):
    MONTHS = zip( range(1,13), range(1,13) )

    tenant = models.ForeignKey(Tenant, null=True, on_delete= models.SET_NULL)
    bill = models.ForeignKey(Bill, null=True, on_delete= models.SET_NULL)
    price = models.FloatField(max_length=100, null=True)
    month = models.IntegerField(choices=MONTHS, blank=True)
    date_created = models.DateTimeField(auto_now_add=True)
    status = models.CharField(max_length=100, default="Čekanje na isplatu")


    def __str__(self):
        return self.bill.name
```

Slika 16. Modeli unutar models.py

Slika 16. prikazuje kreirane modele unutar datoteke *models.py* i neke od spomenutih metoda i relacija objašnjenih u ovom i prethodnim poglavljima.

Pošto kreiranjem ovakvog tipa modela koji su povezani s drugim modelima, potrebno je definirati kako reagiraju kad se jedan od njih izbriše i koju će vrijednost nositi. Zbog toga navesti ćemo bitan parametar koji moramo unijeti u relacijsko polje naziva `on_delete`:

`-on_delete = models.DO_NOTHING` - je najgora opcija zbog toga što se ne izvršava nikakva promjena

`-on_delete = models.SET_NULL` – postavlja null vrijednost kad se izbriše relacijska veza

`-on_delete = models.CASCADE` – uobičajeno je postavljena i odmah briše sve relacijski povezane modele

`-on_delete = models.SET_DEFAULT` – postavlja default vrijednost kada se povezani model izbriše

`-on_delete = models.SET()` – slična metoda kao navedena default ali uz razliku što omogućava postavljanje vrijednosti stranoga ključa vrijednosti proslijeđenoj u set-u ili poziva zadani argument ako je moguće

`-on_delete = models.PROTECT` – blokira brisanje povezanoga modela

4.4. Forme

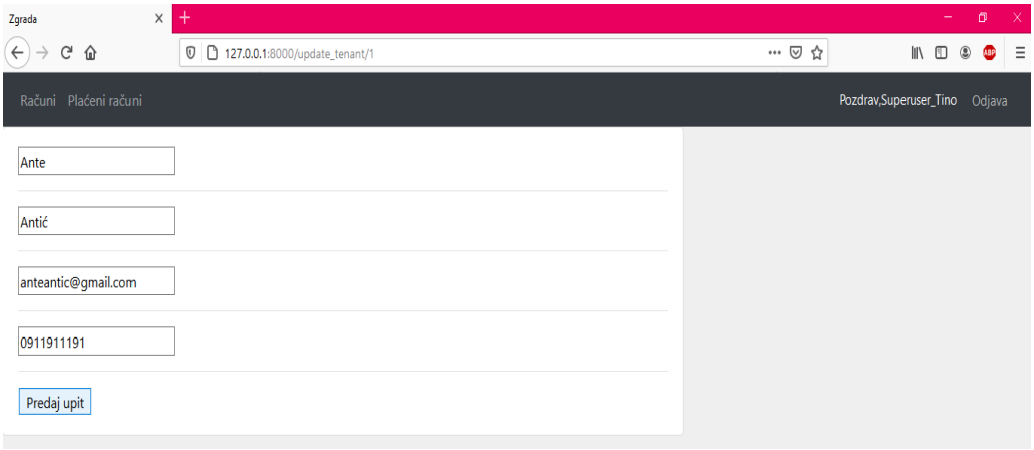
Forme predstavljaju okvire u kojima korisnik unosi tekst, znamenke, cijene i na bilo koji način preko tih okvira upravlja podacima iz baze podataka koji su zadani atributima unutar datoteke models.py. HTML formu definiramo s kôdom `<form>` i zatvaramo ih sa `</form>`.

Za rad Djanga s formama unutar datoteke aplikacije potrebno je kreirati novu datoteku naziva *forms.py*. Unutar te mape se razvijaju željene forme koje se prosljeđuju HTML datoteci u kojoj formu želimo kreirati za prikaz na sučelju. U *froms.py* prvo se na začelju uvoze željeni modeli. Forme kreiramo na sličan način poput modela, s kôdom `class Ime_modelaForm(ModelForm)` i ako su samo potrebni okviri određenoga modela koji se uvozi, kreira se potklasa kôda `class Meta:` ispod koje se navodi model iz kojega se podaci uvoze `model = Ime_modela` i atribute koje će forma obuhvaćati iz modela `fields = ['atribut1', 'atribut2'].` Ako se obuhvaćaju svi podaci upisujemo `'__all__'` u polje `fields.` Ako nam je potrebno da forma sadrži tekst unutar okvira prije potklase, upisuje se ime atributa unutar kojega će pisati tekst i to preko kôda `atribut1 = forms.CharField(max_length=100, help_text=" Unesite tekst „).` U *views.py* definiramo na koji način i kad će se prikazivati forma. Uobičajeno će se prikazivati metodom POST jer većinom se forme koriste za ispunjavanje željene radnje a ako se želi prikazati atribut modela koristi se metoda GET.

```
from django.forms import ModelForm
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User
from django import forms


from .models import *

class TenantForm(ModelForm):
    class Meta:
        model = Tenant
        fields = '__all__'
        exclude = ['user', 'id_tenant']


class PaymentForm(ModelForm):
    class Meta:
        model = Payment
        fields = '__all__'
        exclude= ['tenant', 'status']


class CreateUserForm(UserCreationForm):
    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']


class UpdateTenantForm(ModelForm):
    class Meta:
        model = Tenant
        fields = ['name', 'last_name', 'email', 'phone']
```

Slika 17. Forme unutar forms.py

Slika 17. prikazuje sve forme kreirane unutar aplikacije. Forme *ModelForm* služe za izvlačenje podataka iz baze kreirane od strane korisnika dok *UserCreationForm* izvlači podatke iz uobičajenog Django modela *User* koji se koristi za registriranje i prijavu korisnika. Forme u slučaju kreirane aplikacije služe za mijenjanje podataka o stanaru i računu uz *User* formu koja služi za autentifikaciju.



Slika 18. Forma za mijenjanje podataka o stanaru

Slika 18. prikazuje formu koja služi za jednostavnu izmjenu podataka o stanaru i uz predani upit se dinamično mijenjaju podaci unutar aplikacije. Na ovu formu administrator ulazi preko stranice prikazane na slici 10.

5. Zaključak

Da bi ovakav golemi projekt bio realiziran u općoj upotrebi potrebno je još mnogo toga za implementirati u informatizaciji da bi sustav bio optimiziran. To podrazumijeva širu raščlambu sustava uz detaljni opis svake grane. Raščlanjivanje dijagrama toka podataka na razine 2 i 3. Izrada dijagonalizirane matrice poslovnoga sustava koja uključuje tok odvijanja procesa nad entitetima preko posloženoga redoslijeda. Ali radi jednostavnosti aplikacije opisani su neki osnovni procesi unutar sustava.

U ovome radu opisana je jednostavnost i praktičnost upotrebe Django okvira u razvoju jednostavne aplikacije i rasčlamba aplikacije na osnovnu metodologiju informacijskih sustava. Uz dobru dokumentaciju na Django službenoj stranici korisnik može razviti dobre temelje za daljnju složeniju aplikaciju. Ovakav tip aplikacije se uz daljnji razvoj može dovesti do projekta spremnog za opću primjenu. Sustav plaćanja bi se tako sveo na jednostavniji i kompaktniji model što bi se moglo primjeniti na područjima velike populacije.

Literatura

[1] Warren Lynch, „Data Flow Diagram Comprehensive Guide with Examples", https://medium.com/@warren2lynch/data-flow-diagram-comprehensive-guide-with-examples-d9858387f25e (posjećeno 16.9.2020)

[2] SmartDraw LLC, „Data Flow Diagram“, https://www.smartdraw.com/data-flow-diagram (posjećeno 3.9. 2020)

[3] ThreatModeler Software Inc., „Process Flow vs. Data Flow Diagrams for Threat Modeling“, https://threatmodeler.com/data-flow-diagrams-process-flow-diagrams (posjećeno 16.9. 2020)

[4] Klipfolio Inc., „From Undone to Done: The Essential Guide to Workflow Diagrams“,https://www.klipfolio.com/blog/guide-to-workflow-diagrams (posjećeno 16.9. 2020)

[5] Christine M. Walsh-Kelly MD, „Flow Chart/Process Flow Diagram“, https://www.med.unc.edu/neurosurgery/files/2018/10/Flow-chart-Process-Flow.pdf (posjećeno 16.9.2020)

[6] Klarin, K, „Programsko inženjerstvo“, https://www.oss.unist.hr/sites/default/files/file_attach/Programsko%20in%C5%BEenjerstvo%20-%20Karmen%20Klarin.pdf (posjećeno 16.9. 2020)

[7] Klarin, K, „Informacijski sustavi“, https://vdocuments.mx/informacijski-sustavi-skripta.html (posjećeno 16.9.2020)

[8] Stack Exchange Inc., „How to filter a reverse related field without additional queries?“, https://stackoverflow.com/questions/63746715/how-to-filter-a-reverse-related-field-without-additional-queries (posjećeno 16.9.2020)

[9] Django Software Foundation, „Cross Site Request Forgery Protection“, https://docs.djangoproject.com/en/3.1/ref/csrf (posjećeno 16.9.2020)

[10] Django Software Foundation, „Meet Django“, https://www.djangoproject.com (posjećeno 16.9.2020)

[11] Django Software Foundation, „Announcing the Django Software Foundation“, https://www.djangoproject.com/weblog/2008/jun/17/foundation (posjećeno 16.9.2020)

[12] CS Odessa Corp., „IDEF0 Diagram – Decomposition structure“, https://www.conceptdraw.com/examples/how-to-draw-functional-decomposition-diagram (posjećeno 16.9.2020)

[13] Django Software Foundation, „Working with forms“, https://docs.djangoproject.com/en/3.1/topics/forms/ (posjećeno 16.9.2020)

[14] Django Software Foundation, „Models“, https://docs.djangoproject.com/en/3.1/topics/db/models/ (posjećeno 16.9.2020)

[15] Django Software Foundation, „Templates“, https://docs.djangoproject.com/en/3.1/topics/templates/ (posjećeno 16.9.2020)