Automatizirana web aplikacija za pisanje ispita na kolegiju Uvod u objektno orijentirano programiranje

Škrlj, Luka

SVEUČILIŠTE U RIJECI

Tehnički fakultet

Preddiplomski sveučilišni studij računarstva

Završni rad

Automatizirana web aplikacija za pisanje ispita na kolegiju Uvod u objektno orijentirano programiranje

Rijeka, srpanj 2022.

Luka Škrlj
0069088022

SVEUČILIŠTE U RIJECI

Tehnički fakultet

Preddiplomski sveučilišni studij računarstva

Završni rad

Automatizirana web aplikacija za pisanje ispita na kolegiju Uvod u objektno orijentirano programiranje

Mentor: izv. prof. dr. sc. Jonatan Lerga

Rijeka, srpanj 2022.

Luka Škrlj
0069088022

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 21. ožujka 2022.

Zavod: **Zavod za računarstvo**
Predmet: **Digitalna logika**
Grana: **2.09.06 programsko inženjerstvo**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Luka Škrlj (0069088022)**
Studij: Preddiplomski sveučilišni studij računarstva

Zadatak: **Automatizirana web aplikacija za pisanje ispita na kolegiju Uvod u objektno orijentirano programiranje / Automated Web Application for Writing Exams in the Course Introduction to Object Oriented Programming**

Opis zadatka:

Potrebno je izraditi Django internetsku aplikacije koja automatizira pisanje kolokvija/završnih ispita na kolegiju Uvod u objektno orijentirano programiranje (i omogućava vježbanje zadataka u programskom jeziku Javi). Aplikacija treba omogućavati pokretanja Junit testova kod predaje studentovog zadatka, izradu efikasnog sustava za upravljanje podacima kod predaje zadataka, omogućavanje autorizacije (student, profesor, superadmin). Također, potrebno je izraditi sučelje za profesore te prikaz podataka o uspješnosti studenta na određenim zadatcima, provjerama i zasebno za svakog studenta, kao i osigurati pouzdanu konekciju tokom predaje zadataka.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Luka Škrlj

Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor:

Izv. prof. dr. sc. Jonatan Lerga

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Kristijan Lenac

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, kolovoz 2022. _____

Ime Prezime

SADRŽAJ

1. UVOD

Kod kolegija Uvod u objektno orijentirano programiranje moguće je automatizirati pisanje provjera znanja i vježbanje zadataka za provjere. U ovom radu je izrađena web aplikacija koja automatizira pisanje provjera te omogućuje studentima samostalno vježbanje zadataka, a administratorima sustava daje uvid o napretku studenta kao i jedinstvenu bazu podataka koja sadrži sve zadatke iz provjere na jednom mjestu. Praktični dio projekta je izrađena u programskom jeziku Python uz podršku Django programskog okvira.

Aplikacija administratorima omogućuje praćenje napreteka studenata kao i lako dodavanje istih u sustav. Administratorsko sučelje omogućuje pregled, uređivanje, stvaranje i brisanje svih modela koji se nalaze unutar baze podataka. Administrator sa najvećim ovlastima može kreirati grupe sa određenim dozvolama ili svakom korisniku dodijeliti posebne dozvole.

Kako se na kolegiju Uvod u objektno orijentirano programiranje pišu teorijske i programske provjere, unutar aplikacije je omogućeno pisanje jedne ili druge vrste provjere. Zadaci provjere koja se bazira na praktičnom dijelu nastave mogu se riješavati na temelju izlaza i ulaza iz programskog zadatka ili na temelju testova koji se prokreću u trenutku kada student preda riješenje zadatka.

Kroz ovaj rad upoznaju se tehnologije i alati koje je potrebno savladati za izradu ove aplikacije, kako postaviti projekt na bilo kojem računalu, programski kod koji je zadužen za automatizirano pokretanje zadatka na poslužitelju, kako je aplikacija integrirana u sustav za upravljanje sadržajem te glavne značajke sustava.

2. TEHNOLOGIJE I ALATI KORIŠTENI U IZRADI PROJEKTA

2.1. Programski jezici i softverski okviri

2.1.1.Python

Python je interpretirani, objektno orijentirani programski jezik visoke razine s dinamičkom semantikom. Njegove visoke razine ugrađenih podatkovnih struktura, u kombinaciji s dinamičkim tipkanjem i dinamičkim vezanjem, čine ga vrlo atraktivnim za brzi razvoj aplikacija, kao i za korištenje kao skriptni jezik za međusobno povezivanje postojećih komponenti. Pythonova jednostavna sintaksa naglašava čitljivost i stoga smanjuje troškove održavanja programa. Također, Python podržava module i pakete, što potiče modularnost programa i ponovnu upotrebu koda. Interpreter i opsežna standardna biblioteka dostupni su u izvornom ili binarnom obliku.

Budući da je Python jednostavan, ciklus uređivanja, testiranja i otklanjanja pogrešaka je brz. Otklanjanje pogrešaka u Python programima je jednostavno: greška ili loš unos nikada neće uzrokovati pogrešku segmentacije. Umjesto toga, kada interpreter otkrije pogrešku, pokreće iznimku. Kada program ne uhvati iznimku, interpreter ispisuje trag stoga. Program za ispravljanje pogrešaka napisan je u samom Pythonu te na razini izvora omogućuje pregled lokalnih i globalnih varijabli, procjenu proizvoljnih izraza, postavljanje prijelomnih točaka, prolazak kroz kod redak po redak i tako dalje. Python sadrži mnoge softverske okvire kao što su Django i Pyramid te mnoge napredne sustave za upravljanje sadržajem kao štosu Plone i django CMS[1, 2].

Standardna biblioteka podržava mnoge internetske protokole i biblioteke:

- Hypertext Markup Language (HTML) i extensible markup language (XML)
- JavaScript Object Notation (JSON)
- Obrada e-pošte
- Podrška za File Transfer Protocol (FTP), Internet Message Access Protocol (IMAP) i

druge internetske protokole

- Sučelje internetske utičnice jednostavno za korištenje
- Request, moćna biblioteka Hypertext Transfer Protocol (HTTP) klijenta
- Beautiful Soup, HTML parser koji se može nositi sa svim vrstama neobičnog HTML-a
- Feedparser za analizu Really Simple Syndication (RSS)/Atom feedova
- Paramiko, implementirajući Secure Shell 2 (SSH2) protokol
- Twisted Python, okvir za asinkrono mrežno programiranje

Verzija Pythona koja je upotrebljena u izradi projekta je 3.8.

2.1.2.Django

Django je besplatni okvir za razvoj web aplikacija otvorenog koda napisan u Pythonu. Koristi se za brz i čist razvoj web aplikacija, a izradili su ga iskusni razvojni programeri kako bi olakšali zadatke koji se ponavljaju. Django iza sebe ima opširnu dokumentaciju i veliku zajednicu koja je pridonijela velikom dijelu programskog koda za Django[3].

Neke značajke koje Django čine idealnim okvirom za razvoj web aplikacija su sljedeće:

- Razvoj aplikacija uz Django je izuzetno brz
- Django ima desetke projekata koji se mogu integrirati za obavljanje uobičajenih zadataka kao što su provjera autentičnosti korisnika, autorizacija i administracija sadržaja.
- Može se koristiti za gotovo sve vrste projekata, od sustava za upravljanje sadržajem, preko aplikacija za e-trgovinu pa do platformi za isporuku na zahtjev.
- Ima podršku za sprječavanje uobičajenih sigurnosnih problema, uključujući krivotvorenje zahtjeva između stranica, skriptiranje između stranica i SQL injekcija.
- Web lokacije mogu se brzo skalirati kako bi se zadovoljili zahtjevi velikog prometa.

Verzija Djanga koja je upotrebljena u izradi rada je 3.1.

2.1.3.Platforma django CMS

Platformu django CMS su izvorno osmislili web programeri frustrirani tehničkim i sigurnosnim ograničenjima drugih sustava. Njegova jezgra olakšava integraciju s drugim softverom i brzo podizanje sustava, dok ga jednostavnost upotrebe čini izvrsnim izborom za upravitelje sadržaja, urednike sadržaja i administratore web stranica[4].

Programerima je omogućeno brzo integriranje druge postojeće Django aplikacije ili izgradnja potpuno novih kompatibilnih aplikacija koje iskorištavaju značajke objavljivanja i uređivanja django CMS-a. Sučelje je jednostavno za korištenje i ima vrlo intuitivan sustav s priključcima koje je moguće povlačiti i spustiti na određena mjesta unutar web stranice. Unutar Content Management Systema (CMS) mogu postojati stranice u višejezičnim verzijama. Platforma django Content Management System-a (CMS) je otvorenog koda i podržava ga zajednica suradnika[5].

Neke ključne značajke django CMS-a su:

- snažna internacionalizacija te podrška za stvaranje višejezičnih stranica
- podrška za razne uređivače s naprednim značajkama za uređivanje teksta
- provjera autentičnosti korisnika
- autorizacija i administracija sadržaja
- fleksibilan sustav dodataka koji programerima omogućuje da moćne alate stave na dohvat ruke uređivačima
- temeljitu dokumentaciju
- jednostavna i sveobuhvatna integracija u postojeće projekte
- sastavljen je od održivog koda, uključujući naglasak na automatizirano testiranje

Tablica 2.1 *Podržane verzije django CMS-a*[5]

django CMS	Python						Django						
	3.9	3.8	3.7	3.6	3.5	3.4	3.2	3.1	3.0	2.2	2.1	2.0	1.11
4.0.x	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✗	✗	✗
3.9.x	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✗	✗	✗
3.8.x	✗	✓	✓	✓	✓	✗	✗	✓	✓	✓	✗	✗	✗
3.7.x	✗	✓	✓	✓	✓	✓	✗	✗	✓	✓	✓	✓	✓
3.6.x	✗	✗	✓	✓	✓	✓	✗	✗	✗	✓	✓	✓	✓
3.5.x	✗	✗	✓	✓	✓	✓	✗	✗	✗	✗	✗	✗	✓
3.4.5	✗	✗	✗	✓	✓	✓	✗	✗	✗	✗	✗	✗	✓

Na tablici 2.1 prikazane su verzije Pythona i Djanga koje django CMS podržava. Kvačica označava da je verzija testirana i funkcionalna, a križić označava da verzija nije ispitana ili je nekompatibilna. Verzije korištene unutar projekta su podcrtane.

2.1.4.Junit

JUnit je okvir za jedinično testiranje za programski jezik Java[6]. Značajke JUnita su:

- JUnit je okvir otvorenog koda koji se koristi za pisanje i izvođenje testova
- pruža bilješke za prepoznavanje metoda ispitivanja
- pruža tvrdnje za testiranje očekivanih rezultata
- pruža pokretače testova za izvođenje testova
- testovi nisu složeni i pokretanje testova je brzo
- jednostavna i sveobuhvatna integracija u postojeće projekte
- testovi se mogu organizirati u testne pakete koji sadrže testne slučajeve

2.1.5. Bootstrap

Bootstrap je besplatan okvir za razvoj web aplikacija i otvorenog je koda. Osmišljen je kako bi

olakšao proces razvoja responzivnih mobilnih internetskih stranica. Okvir omogućuje da se brže izgrade web stranice jer nije potrebno brinuti o detaljima.

Sastoji se od skripti temeljenih na HTML-u, Cascading Style Sheets (CSS)-u i JavaScriptu sa kojima su izrađene razne komponente. Isto tako Bootstrap osigurava da svi elementi sučelja web stranice rade optimalno na svim veličinama zaslona. Verzija Bootstrapa s "izvornim kodom" omogućuje pristup Sass priključku. To znači da stvara prilagođenu tablicu stilova koja uvozi Bootstrap, omogućujući da se po potrebi izmijeni i proširi alat.

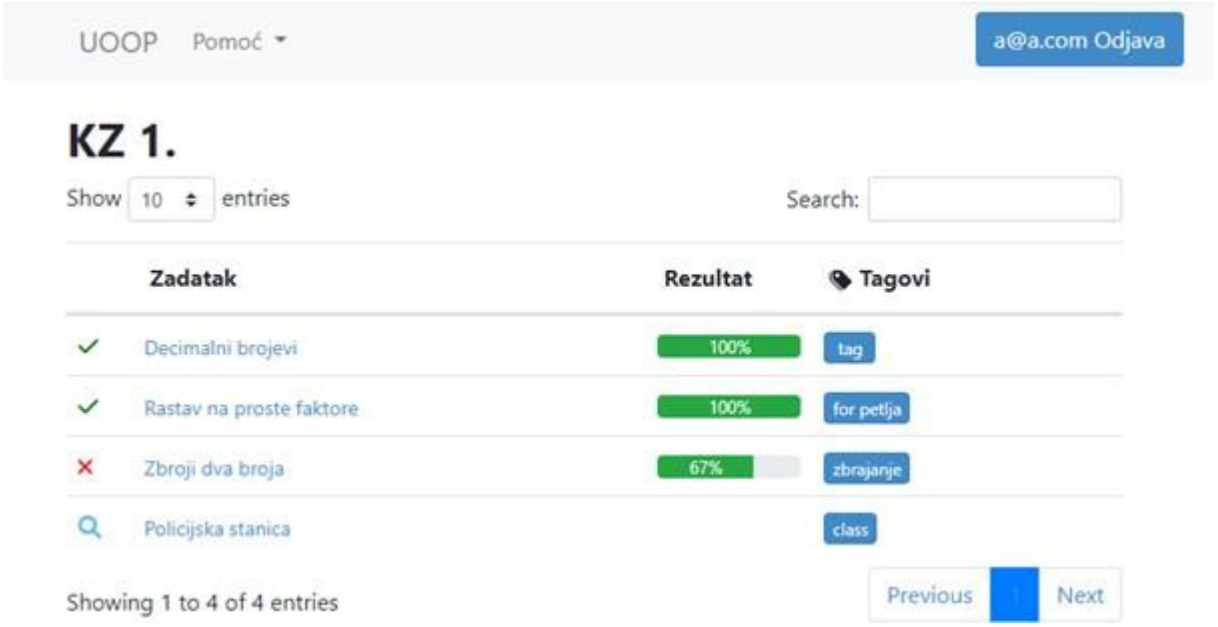
Bootstrap se može instalirati i s upraviteljem paketa. Upravitelj paketa je alat koji upravlja i ažurira okvire, biblioteke i module. Neki od najpopularnijih upravitelja paketima uključuju npm, Composer i Bower. Composer upravlja ovisnostima na strani poslužitelja, dok se npm fokusira na korisničku stranu.

Bootstrap zahvaljujući svojoj popularnosti ima veliku zajednicu u kojoj se raspravlja kako ga unaprijediti i popraviti postojeće nedostatke kod komponenta. Neke od komponenti sučelja Bootstrapa uključuju navigacijske trake, mrežne sustave, karusele slika i gumbe. Velika prednost je to što je Bootstrap lako naučiti. Jedan od razloga zašto je Bootstrap tako popularan među web programerima i web dizajnerima je taj što ima jednostavnu strukturu datoteka. Datoteke su kompajlirane za jednostavan pristup, a za njihovu izmjenu potrebno je samo osnovno poznavanje HTML-a, CSS-a i JavaScript-a.

Kako bi se smanjilo vrijeme učitavanja stranice, Bootstrap minimizira CSS i JavaScript datoteke. Osim toga, Bootstrap održava dosljednost u sintaksi između web stranica i programera, što je idealno za timske projekte [7].

```
<table
    id="example"
    class="table /*table-striped*/ /*table-bordered*/"
>
```

U primjeru je prikazana jednostavna HTML oznaka tablice. Klase *table*, *table-striped* i *table-bordered* su Bootstrapove klase. One definiraju ponašanje tablice.



Slika 2.1 *Izgled tablice*

Na slici 2.1 vidimo da je omogućena paginacija i pretraživanje elemenata unutar tablice koje su omogućile klase *table*, *table-striped* i *table-bordered*.

2.2. Paketi i moduli

2.2.1. Virtualenv

Python aplikacije će često koristiti pakete i module koji ne dolaze kao dio standardne biblioteke. Aplikacije će ponekad trebati određenu verziju biblioteke, jer aplikacija može zahtijevati ispravljanje određene pogreške ili aplikacija može biti napisana korištenjem zastarjele verzije sučelja biblioteke. To znači da možda neće biti moguće da jedna instalacija Pythona ispuni zahtjeve svake aplikacije. Ako aplikacija A treba verziju 1.0 određenog modula, ali aplikacija B treba verziju 2.0, tada su zahtjevi u sukobu i instaliranje verzije 1.0 ili 2.0 neće moći pokrenuti jednu aplikaciju.

Rješenje za ovaj problem je stvaranje virtualnog okruženja, samostalnog stabla direktorija koje sadrži Python instalaciju za određenu verziju Pythona, plus niz dodatnih paketa. Različite aplikacije tada mogu koristiti različita virtualna okruženja. Za rješavanje ranijeg primjera proturječnih zahtjeva, aplikacija A može imati svoje vlastito virtualno okruženje s instaliranom verzijom 1.0 dok aplikacija B ima drugo virtualno okruženje s verzijom 2.0. Ako aplikacija B zahtijeva nadogradnju biblioteke na verziju 3.0, to neće utjecati na okruženje aplikacije A.

Modul koji se koristi za stvaranje i upravljanje virtualnim okruženjima zove se venv. Taj modul će obično instalirati najnoviju verziju Pythona koja je dostupna na računalu. Ako na računalu postoji više verzija Pythona, modul omogućuje odabir određene verzije Pythona. Da bi se stvorilo virtualno okruženje potrebno je nalaziti se unutar željenog direktorija i pokreniti *venv* modul kao skriptu sa stazom direktorija to jest pokrenuti naredbu *python3 -m venv uoop*. Ovo će stvoriti direktorij *uoop* ako ne postoji, a također će stvoriti direktorije unutar njega koji sadrže kopiju Python interpretera i razne prateće datoteke. Uobičajena lokacija direktorija za virtualno okruženje je *.venv*. Ovo ime je obično skriveno u ljusci i stoga nije važno koji je naziv okruženja dodan. Također, virtualno okruženje sprječava sukob s definiranim varijablama unutar *.env* datoteke [8].

2.2.2.Paket Django-environ

Ideja ovog paketa jest objediniti mnogo paketa koji rade iste stvari. Na primjer niz iz *os.environ* se može raščlaniti i pretvoriti u neke od korisnih varijabli unesenih u Python. Da bi se to postiglo neki nizovi veze su izraženi kao url, tako da ga ovaj paket može analizirati. Nizovi iz *os.environ* učitavaju se iz *.env* datoteke i popunjavaju u *os.environ* metodom *setdefault,* kako bi se izbjeglo brisanje stvarnog okruženja. Ovaj paket je vrlo koristan jer je važno držati osjetljive dijelove koda poput Application Programming Interface (API) ključeva i lozinki.

```
import environ
env = environ.Env() environ.Env.read_env()
```

Za koristiti modul potrebno ga je uvesti sa *import environ*, kao što je prikazano u isječku koda, te nakon toga instancirati *Env* klasu čiji konstruktor prima naziv varijable pohranjene unutar *.env* datoteke. Nakon instanciranja klase pozvana je statička funkcija *read_env()* koja učitava sve varijable spremljene unutar *.env* datoteke. Nakon inicijalizacije, *env()* funkcija može se koristiti za dohvat varijabli spremljenih u *.env* datoteci. Unutar praktičnog dijela projekta *.env* datoteka čuva sve podatke koji su potrebni za spajanje s bazom podataka[9].

2.2.3.Python-Markdown

Markdown je alat za pretvorbu teksta u HTML i omogućuje pisanje koristeći format običnog teksta koji je jednostavan za čitanje i pisanje, a zatim se tekst pretvori u strukturirani HTML. Pojam "Markdkown" označava dvije stvari:

- sintaksa oblikovanja običnog teksta
- softverski alat, napisan u Perlu, koji pretvara format običnog teksta u HTML

Najvažniji cilj dizajna sintakse oblikovanja Markdowna je učiniti tekst što čitljivijim. Ideja je da dokument formatiran Markdownom treba biti moguće objaviti takav kakav jest, kao običan tekst, a da ne izgleda kao da je označen oznakama ili uputama za oblikovanje. Iako je sintaksa Markdowna pod utjecajem nekoliko postojećih filtara pretvaranja teksta u HTML, najveći pojedinačni izvor inspiracije za Markdownovu sintaksu je format e-pošte s običnim tekstom[10].

Uz osnovnu sintaksu označavanja iz Markdown softverskog alata, Python-Markdown podržava sljedeće značajke:

- prihvaća unos na bilo kojem jeziku koji podržava Unicode uključujući dvosmjerni tekst
- dostupna su različita proširenja za promjenu i/ili proširenje osnovne sintakse
- dostupan je javni API za proširenja za pisanje vlastitih proširenja
- Izlazni format može ispisati dokumente s HTML ili XHTML stilskim oznakama[11]

2.3. Ostali alati

2.3.1. Visual Studio Code

Visual Studio Code besplatni je uređivač koda koji ubrzava proces kodiranja. Uređivač podržava sve programske jezike i omogućuje korištenje mnogo različitih kratica na tipkovnici. Tijekom kodiranja, Visual Studio Code daje prijedloge za dovršavanje redaka koda i brze popravke za uobičajene pogreške. Također može se koristiti program za ispravljanje pogrešaka koji prolazi kroz svaku liniju koda i automatski formatira i ispravlja kod. Dostupno je mnogo ekstenzija za Visual Studio Code, a tijekom kodiranja projekta su korištene Django i Python ekstenzije koje omogućuju ispravke i daju boju kodu kako bi se brže uočile logičke i sintaktičke greške[12].

2.3.2. MySQL Workbench

MySQL Workbench je objedinjeni vizualni alat za arhitekte baza podataka, programere i administratore baza podataka. Taj alat omogućuje modeliranje podataka, razvoj Structured Query Language (SQL)-a i opsežne administrativne alate za konfiguraciju poslužitelja, korisničku administraciju, sigurnosno kopiranje i još mnogo toga. Dostupan je u sustavima Windows, Linux i Mac OS X. Omogućuje laku analizu baze podataka programeru ili arhitektu podataka kao i vizualni dizajn, modeliranje, generiranje i upravljanje bazama podataka. Uključuje sve što arhitektu baze podataka treba za stvaranje složenih ER modela. MySQL Workbench isporučuje vizualne alate za kreiranje, izvršavanje i optimiziranje SQL upita. SQL Editor omogućuje označavanje sintakse u boji, automatsko dovršavanje, ponovnu upotrebu SQL isječaka i povijest izvršavanja SQL-a. Ploča povezivanja baze podataka omogućuje

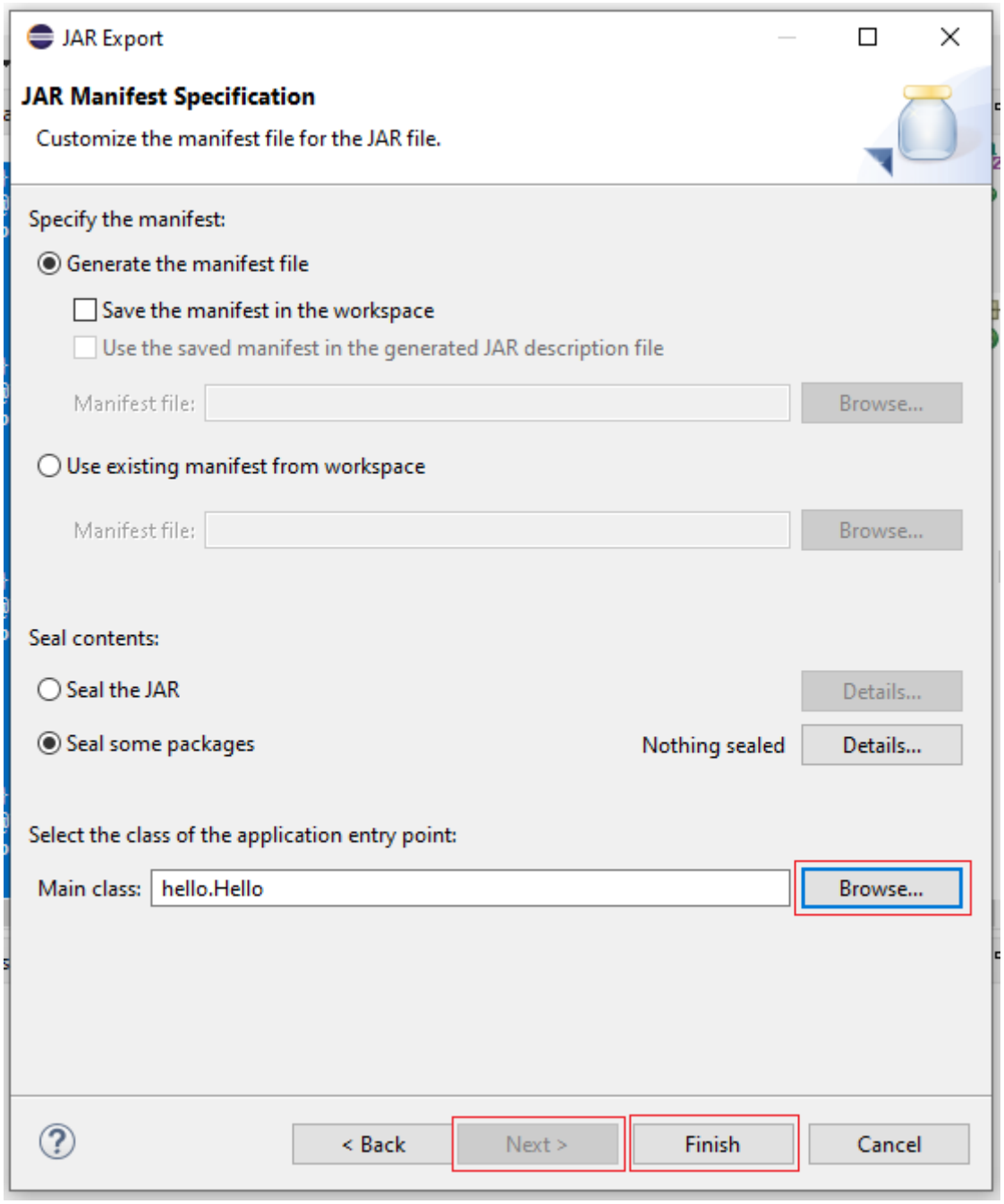
programerima jednostavno upravljanje standardnim vezama baze podataka. Preglednik objekata omogućuje trenutni pristup shemi baze podataka i objektima. Programeri i administratori baze podataka mogu koristiti vizualne alate za konfiguriranje poslužitelja, upravljanje pravima korisnika, izvođenje sigurnosne kopije i oporavka, pregled revizijskih podataka i pregled stanja baze podataka[13].

2.3.3.Git

Git je besplatni distribuirani sustav za kontrolu verzija otvorenog koda koji je osmišljen za brzu i učinkovitu obradu svih veličina projekata. Alat je jednostavan za naučiti i ima brzu izvedbu, mogućnost lokalnog grananja, prikladna pripremna područja i višestruki tijek rada. Postoje mnoge platforme koje su bazirane na Git-u od kojih su najpopularnije GitLab i GitHub, a za razvoj ovog projekta korišten je GitHub. GitHub je platforma namijenjena za pregled koda, kontrolu verzija i suradnju te omogućuje programerima da zajedno rade na projektima s bilo kojeg mjesta[14].

2.3.4.Eclipse

Eclipse je integrirano razvojno okruženje za razvoj aplikacija korištenjem programskog jezika Java i drugih programskih jezika[15]. U sklopu ovog projekta Eclipse se koristi za pisanje zadataka i JUnit testova sa kojima se testira funkcionalnost aplikacije. Eclipse omogućuje lako baratanje s Java ARchiven (JAR) datotekama i pokretanje JUnit testova. Kako bi zapakirali JAR datoteku uz pomoć Eclipse uređivača, potrebno je kliknuti desni klik na željeni projekt unutar stabla projekta na lijevoj strani ekrana pa potom na stavku „Export...“. Potom će se otvoriti dijaloški okvir unutar kojeg je pod stavkom Java potrebno odabrati „JAR file“ nakon čega se otvara novi dijaloški okvir. Na tom okviru je potrebno odabrati klasu unutar koje se nalazi metoda *main().* Ta postavka je važna jer JAR datoteke neće imati zabilježen put do početne točke programa pa se program neće izvršiti, no JUnit testovi se mogu pokrenuti bez obzira na tu postavku.



Slika 2.2 *Dijaloški okvir za kreiranje JAR datoteke*

3. DJANGO SINTAKSA

Django naglašava mogućnost ponovne upotrebe komponenti, koja se također naziva Don't Repeat Yourself (DRY) i dolazi sa značajkama spremnim za korištenje kao što su sustav za prijavu, veza s bazom podataka i Create Read Update Delete (CRUD) operacije.

Django slijedi Model View Template (MVT) obrazac dizajna:

- Model - Podaci koji su predstavljeni, obično podaci iz baze podataka.
- Pogled - Rukovatelj zahtjevima koji vraća relevantni predložak i sadržaj na temelju zahtjeva korisnika.
- Predložak - tekstualna datoteka (poput HTML datoteke) koja sadrži izgled web stranice, s logikom prikaza podataka.

3.1. Pogledi

Django pogledi ključna su komponenta aplikacija izgrađenih s Djangom. U svom najjednostavnijem obliku oni su Python funkcija ili klasa koja preuzima web zahtjev i vraća web odgovor. Pogledi se koriste za obavljanje stvari kao što su dohvaćanje objekata iz baze podataka, mijenjanje tih objekata ako je potrebno, renderiranje obrazaca, vraćanje HTML-a i još mnogo toga.

Django ima dvije vrste pogleda; pogledi temeljeni na funkcijama i pogledi temeljeni na klasama. Django je izvorno započeo samo s pogledima temeljenim na funkcijama, ali je zatim dodao poglede temeljene na klasama kao način za apstrakciju funkcionalnosti. Pogledi temeljeni na klasama povećavaju produktivnost kada nije potrebna nikakva posebna implementacija funkcionalnosti na određenim zahtjevima.

U svojoj srži, pogledi temeljeni na klasama su Python klase. Django se isporučuje s raznim predlošcima koji imaju unaprijed konfiguriranu funkcionalnost koja se može ponovno koristiti i često proširiti. Tim klasama zatim se daju korisna imena koja opisuju koju vrstu funkcionalnosti

pružaju. Često se oni nazivaju "generički pogledi" jer pružaju rješenja za uobičajene zahtjeve. Ovaj projekt koristi poglede temeljene na funkcijama jer takvi pogledi ne zahtijevaju nadjačavanje metoda i imaju više mogućnosti[16].

```
def course(request, id):
course = Course.objects.get(id=id)
tests= Test.objects.filter(course=id)quizs = Quiz.objects.filter(
course=id)return render(request, 'course.html',
{'course': course, 'tests': tests, 'quizs': quizs})
```

Isječak koda prikazuje pogled temeljen na funkcijama. Pogled kao parametre prima „request“ i „id“ koji predstavljaju zahtjev i primarni ključ koji je učitan iz niza znakova unutar Uniform Resource Locator-a (URL-a) koji okida ovaj pogled. U ovom slučaju „id“ će biti primarni ključ kolegija kojeg je korisnik odabrao. Nakon što se pogled okine iz baze podataka se dohvaćaju informacije o odabranom kolegiju, svim teorijskim testovima i testovima s programskim zadacima.

Funkcija *get()* dohvaća informacije o kolegiju na vrijednosti njegovog primarnog ključa, a funkcija *filter()* dohvaća sve testove koji su povezani s tim kolegijem. Pogled vraća funkciju *render()* koja kombinira zadani predložak sa zadanim kontekstom i vraća objekt *HttpResponse* s tim prikazanim tekstom. Obavezni argumentisu zahtjev na temelju kojeg će se generirati odgovor i ime predloška koji se treba prikazati. Opcionalni argument je kontekst objekt koji dopušta da se u predlošku koriste vrijednosti postavljene unutar objekta.

3.2. Predlošci

Django treba prikladan način za dinamičko generiranje HTML-a. Najčešći pristup oslanja se na predloške. Predložak je tekstualni dokument ili niz označen pomoću Djangovog jezika predloška. Neki dijelovi predloška su prepoznati i interpretirani od strane sustava koji upravlja predlošcima. Glavni dijelovi predloška su varijable i oznake. Renderiranje zamjenjuje varijable s njihovim vrijednostima, koje se nalaze u kontekstu predloška. Sve ostalo se prikazuje kako je specificirano sa HTML-om. Predložak sadrži statične dijelove željenog HTML izlaza kao i

posebnu sintaksu koja opisuje kako će se umetnuti dinamički sadržaj. Projekt se može konfigurirati s jednim ili nekoliko mehanizama za predloške (ili čak niti jedan ako se predlošci ne koriste).

Django definira standardni API za učitavanje i prikazivanje predložaka bez obzira na pozadinu. Učitavanje se sastoji od pronalaženja predloška za dati identifikator i obrade predloška. Obrada znači interpolaciju predloška s podacima o kontekstu i vraćanje rezultirajućeg niza. Jezik predložaka Django je Djangov vlastiti sustav predložaka. Neke Djangove standardne aplikacije uključuju predloške, poput *djansgo.contrib.admin.* koji u sebi sadrži predloške za prikaz administracijskog sučelja[17].

3.3. Modeli

Model je jedini, konačni izvor informacija o podacima. Sadrži bitna polja i opisuje ponašanje podataka. Općenito, svaki se model preslikava u jednu tablicu baze podataka. Svaki model je Python klasa koja je podklasa *django.db.models.Model*. Svaki atribut modela predstavlja polje baze podataka. Uz modele Django pruža API za pristup bazi podataka. Neki nazivi atributa su zabranjeni poput *clean, save* ili *delete* jer se sukobljavaju s metodama iz klase *Model*.

```
class Test(models.Model):
        title=models.CharField(max_length=50)
        course = models.ForeignKey("Course",on_delete=models.CASCADE)
        startDate = models.DateTimeField()
        endDate = models.DateTimeField()

        def __str__(self):
            return self.title
```

Na isječku koda je prikazan izgled modela *Test*. Metoda *__str__()* poziva se kad god se poziva *str()* na objektu koji nasljeđuje *Model*. Django koristi *str(obj)* na više mjesta, prije svega, za prikaz objekta na Django administratorskoj stranici i kao vrijednost umetnuta u predložak kada prikazuje objekt. Stoga je potrebno vratiti čitljivu reprezentaciju modela iz metode *__str__()*.

Model prikazan u isječku koda će stvoriti tablicu u bazi podataka kao da u MySql-u upišemo kod sa isječka.

```
CREATE TABLE `app_test` (
    `id` int NOT NULL AUTO_INCREMENT,
    `title` varchar(50) NOT NULL,
    `startDate`          datetime(6) NOT NULL, `endDate` datetime(6) NOT
    NULL,
    `course_id` int NOT
    NULL, PRIMARY KEY (`id`
    ),
    KEY `
    app_test_course_id_8c1a8a1d_fk_app_course_id`(`
    course_id`),
    CONSTRAINT `
    app_test_course_id_8c1a8a1d_fk_app_course_id` FOREIGN
    KEY (`course_id`) REFERENCES `app_course`(`id`)
) ENGINE=InnoDB AUTO_INCREMENT=2
DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Za definiranje odnosa više na jedan koristiti se *django.db.models.ForeignKey*. ForeignKey zahtijeva kao argument klasu na koju je model povezan i *on_delete* parametar koji specificira koju vrijednost će poprimiti atribut u tablici prilikom brisanja zapisa. U primjeru je korišten *CASCADE* koji specificira da se prilikom brisanja objekta brišu objekti koji imaju reference na njega. U isječku koda vidljivoje da ako obrišemo jedan kolegij, obrisati će se i svi ispiti koji su na njega povezani.

Django automatski izvodi naziv tablice baze podataka iz naziva klase modela i aplikacije koja ga sadrži. Naziv tablice baze podataka modela konstruira se spajanjem naziva koji se koristio prilikom pokretanja naredbe za stvaranje nove aplikacije *manage.py startapp* s nazivom klase modela između njih. U isječku koda iznad ime tablice glasi *app_test* gdje *app* predstavlja naziv aplikacije, a *test* predstavlja naziv modela. Naziv tablice unutar baze podataka je moguće

nadjačati sa upotrebom parametra *db_table* unutar klase *Meta*. Ako je naziv tablice baze podataka rezervirana riječ za SQL ili sadrži znakove koji nisu dopušteni u nazivima varijabli Pythona, posebice crticu, to je u redu. Django citira nazive stupaca i tablica.

Prema zadanim postavkama, Django daje svakom modelu primarni ključ koji se automatski povećava s vrstom navedenom po aplikaciji u *AppConfig.default_auto_field* ili globalno u postavci *DEFAULT_AUTO_FIELD*. Ako je potrebno navesti prilagođeni primarni ključ, može se navesti *primary_key=True* u jednom od polja. Svaki model zahtijeva da točno jedno polje ima *primary_key=True* (bilo eksplicitno deklarirano ili automatski dodano). Nakon konstruiranja modela Django treba znati kako doći do datoteke unutar koje se nalaze modeli. Unutar datoteke u kojoj se definiraju postavke se nalazi postavka *INSTALLED_APPS* unutar koje se mora nalaziti naziv modula koji sadrži *models.py* datoteku unutar koje su kreirani modeli.

INSTALLED_APPS je popis nizova koji označavaju sve aplikacije koje su omogućene unutar Django instalacije. Svaki niz koji se nalazi unutar liste treba biti Python staza koja vodi do konfiguracije aplikacije ili paket koji sadrži aplikaciju. Kada nekoliko aplikacija pruža različite verzije istog resursa (predložak, statička datoteka, upravljačka naredba, prijevod), prednost ima aplikacija navedena prva u *INSTALLED_APPS* pa je poželjno da svaka aplikacija ima različite nazive resursa. Nakon što je nova aplikacija definirana unutar *INSTALLED_APPS*, potrebno je pokrenuti *python manage.py makemigrations* koji generira migracije na temelju modela koji se nalaze unutar datoteke *models.py*. Migracijske datoteke je potrebno migrirati naredbom *python manage.py migrate* nakon koje se stvara baza podataka opisana u migracijskim datotekama[18].

3.4. Forme

Django nudi niz alata i biblioteka koje su vrlo korisne u izradi obrazaca za prihvaćanje unosa od posjetitelja stranice, a zatim obradu i odgovor na unos.

U HTML-u, obrazac je zbirka elemenata unutar <*form*>...</*form*> koji omogućuju posjetitelju da radi stvari poput unosa teksta, odabira opcija, manipuliranja objektima ili kontrolama i tako dalje, a zatim šalje te informacije natrag na poslužitelj. Neki od ovih elemenata sučelja obrazaca, unos teksta ili potvrdni okviri, ugrađeni su u sam HTML. Sučelje koje sadrži birač datuma ili

omogućuje pomicanje klizača ili upravljanje kontrolama obično će koristiti JavaScript i CSS kao i HTML <*input*> elemente za prikaz tih komponenta.

Element <*input*> mora specificirati URL na koji se trebaju vratiti podaci koji odgovaraju korisnikovom unosu i HTTP metodu kojom se podaci trebaju vratiti. Kao primjer, obrazac za prijavu za Django administratora sadrži nekoliko <*input*> elemenata: jedan od *type="text"* za korisničko ime, jedan od *type="password"* za lozinku i jedan od *type="submit"* za gumb „Prijava“. Također sadrži neka skrivena tekstualna polja koja korisnik ne vidi, a koja Django koristi kako bi odredio što dalje učiniti. Također govori pregledniku da se podaci obrasca trebaju poslati na URL naveden u atributu akcije <*form*> (*/admin/*) i da se trebaju poslati pomoću HTTP mehanizma određenog atributom metode. Kada se klikne na element <*input type="submit" value="Log in"*>, podaci se vraćaju u */admin/*. *GET* i *POST* jedine su HTTP metode koje se koriste kada se radi s obrascima i obično se koriste za različite svrhe. Djangov obrazac za prijavu vraća se metodom *POST*, u kojoj preglednik grupira podatke obrasca, kodira ih za prijenos, šalje poslužitelju i zatim prima natrag svoj odgovor. Nasuprot *POST* metodi, *GET* metoda spaja poslane podatke u niz i koristi ga za sastavljanje URL-a. URL sadrži adresu na koju se podaci moraju poslati, kao i podatkovne ključeve i vrijednosti.

Svaki zahtjev koji bi se mogao koristiti za promjenu stanja sustava, na primjer, zahtjev koji vrši promjene u bazi podataka trebao bi koristiti *POST* metodu. *GET* metodu treba koristiti samo za zahtjeve koji ne utječu na stanje sustava. *GET* metoda također ne bi bila prikladna za obrazac za lozinku jer bi se lozinka pojavila u URL-u, a time i u povijesti preglednika i zapisnicima poslužitelja i to u običnom tekstu. Ne bi bio prikladan ni za velike količine podataka ili za binarne podatke, poput slike. Web aplikacija koja koristi *GET* metode za administratorske obrasce predstavlja sigurnosni rizik: napadaču može biti lako oponašati zahtjev obrasca za pristup osjetljivim dijelovima sustava. *POST*, u kombinaciji s drugim zaštitama poput Djangove Cross-Site Request Forgery (CSRF) zaštite, nudi više kontrole nad pristupom. S druge strane, *GET* je prikladan za stvari kao što je obrazac za pretraživanje weba, jer se URL-ovi koji predstavljaju *GET* zahtjev mogu lako označiti, podijeliti ili ponovno poslati. U središtu ovog sustava komponenti je Djangova klasa *Form*. Na sličan način na koji Django model opisuje logičku strukturu objekta, njegovo ponašanje i način na koji su nam njegovi dijelovi predstavljeni, klasa *Form* opisuje formu i određuje kako ona radi i kako se pojavljuje.

Na sličan način na koji se polja klase modela preslikavaju u polja baze podataka, polja klase obrasca preslikavaju se u <*input*>elemente HTML obrasca (*ModelForm* preslikava polja klase modela u <*input*> elemente HTML obrasca putem obrasca te na tome se temelji Django administracijsko sučelje.)[19].

Polja obrasca su sama po sebi klase, oni upravljaju podacima obrasca i provode provjeru valjanosti kada se obrazac pošalje. Polja *DateField* i *FileField* obrađuju vrlo različite vrste podataka i moraju raditi različite stvari s njima. Polje obrasca predstavlja se korisniku u pregledniku kao HTML "*widget*". Svaki tip polja ima odgovarajuću zadanu klasu *widgeta*, ali one se prema potrebi mogu nadjačati. Za renderiranje objekta u Djangu potrebno je u pogledu dohvatiti objekt iz baze podataka, proslijedite ga u kontekst predloška i proširit ga HTML oznakama pomoću varijabli predloška. Prikazivanje obrasca u predlošku uključuje gotovo isti posao kao i prikazivanje bilo koje druge vrste objekta. Kada se instancira obrazac, može se ostaviti prazan ili se može unaprijed popuniti, na primjer:

- podacima iz spremljene instance modela (kao u slučaju administratorskih obrazaca za uređivanje)
- podacima koji su prikupljeni iz drugih izvora
- podacima primljenim iz prethodnog slanja HTML obrasca

```
class AssignmentForm (forms.ModelForm) :

    class Meta :
        model = UserAssignment
        fields = ('jar', )
```

Na isječku koda je prikazana klasa *AssignmentForm* koja nasljeđuje klasu *ModelForm*. *ModelForm* gradi formu na temelju definiranog modela. Model i polja koja će se prikazati na formi se definiraju unutar klase *Meta*. Budući da je korisniku potrebno omogućiti pristup samo JAR datoteci koju predaje, pod *fields* je dodano samo 'jar' polje koje bi trebalo sadržavati JAR datoteku. Forma će se instancirati kada korisnik klikne na zadatak iz testa.

```
path (" assignment/<int:id >/", views . assignment name="assignment ") ,
```

Klikom na zadatak mijenja se ruta u rutu koja je prikazana u isječku koda i šalje se zahtjev na pogled *assignment() GET* metodom unutar kojeg se dohvaća zapis *UserAssignment* kao što je prikazano u sljedećem isječku koda.

```
user Assignment = User Assignment . o b j e c t s . f i l t e r (
        assignment=assignment . id , newuser=r e q u e s t . user . id
    ) . f i r s t ()
```

Zapis se dohvaća na temelju parametra „id“ koji se proslijedio preko URL-a i na temelju prijavljenog korisnika čiji se podaci nalaze unutar zahtjeva. Klikom na gumb „Predaj“ korisnik predaje formu i šalje se zahtjev sa *POST* metodom. Budući da je korisniku potrebno prikazati datoteku koju je predao, potrebno je stvoriti novi zapis u slučaju da korisnik još nije predao rješenje zadatka. Ako je forma dohvaćena metodom *POST* ili *GET* (u slučaju da je korisnik predao zadatak ili samo kliknuo na zadatak) u njoj će se nalaziti datoteka koja je spremljena unutar *media* direktorija. U slučaju da takav zapis ne postoji, prikazat će se prazna forma.

```
i f  r e q u e s t . method ==  'POST' :
        i f ( user Assignment  i  s  None  )  :
            user Assignment  = User Assignment (
                assignment=context [ ' assignment ' ] ,
                newuser=r e q u e s t . user )
            user Assignment . save ()


        form  =  AssignmentForm ( r e q u e s t .POST,  r e q u e s t . FILES ,
        i n s t a n c e=user Assignment )
        i f  form . is__v a l i d ()
            : form . save ()

        . . .

        context [ ' form ' ] =  form
```

```
return render(request, 'assignment.html', context)
```

Sa isječka je vidljivo da će se forma spremiti u slučaju da je validna jer metoda *is_valid()* provjerava validnost forme i samo u slučaju ako je forma validna poziva se metoda *save()*. Forma se na dnu pogleda postavlja u *context* objekt i prosljeđuje u predložak *assignment.html* kao što je prikazano u isječku koda iznad teksta. Unutar predloška forma se prikazuje kombinacijom HTMLA, CSS-a, JavaScripta i sintakse predloška. Da bi se polja unutar forme prikazivala jedno ispod drugog forma sadržava dodatak *as_p* koji odvaja polje HTML elementima paragrafa.

```
<form id="form-uploadJar" onsubmit="JARsubmited()" method="post"
enctype="multipart/form-data"
  style="margin-top: 2%">
      {% csrf_token %}
      {{ form.as_p }}
      <button type="submit" class="btn btn-primary">
          Predaj
      </button>
      {% if assignment.assignmentTemplate %}
          <a class="btn btn-light"href="{%
              url 'app:download_template' assignment.id
          %}">Download template</a>
      {% endif %}
  </form>
```

Kako bi se forma mogla predati potrebno je proslijediti i CSRF token koji je obilježen na isječku koda oznakom *{% csrf_token %}*. CSRF oznaka predloška pruža zaštitu od krivotvorenja zahtjeva na različitim stranicama. Ova vrsta napada događa se kada zlonamjerno web mjesto sadrži poveznicu, gumb obrasca ili neki JavaScript koji je namijenjen za izvođenje neke radnje na web mjestu, koristeći vjerodajnice prijavljenog korisnika koji posjećuje zlonamjerno mjesto u svom pregledniku. Zaštita je aktivirana prema zadanim postavkama u *MIDDLEWARE* postavci

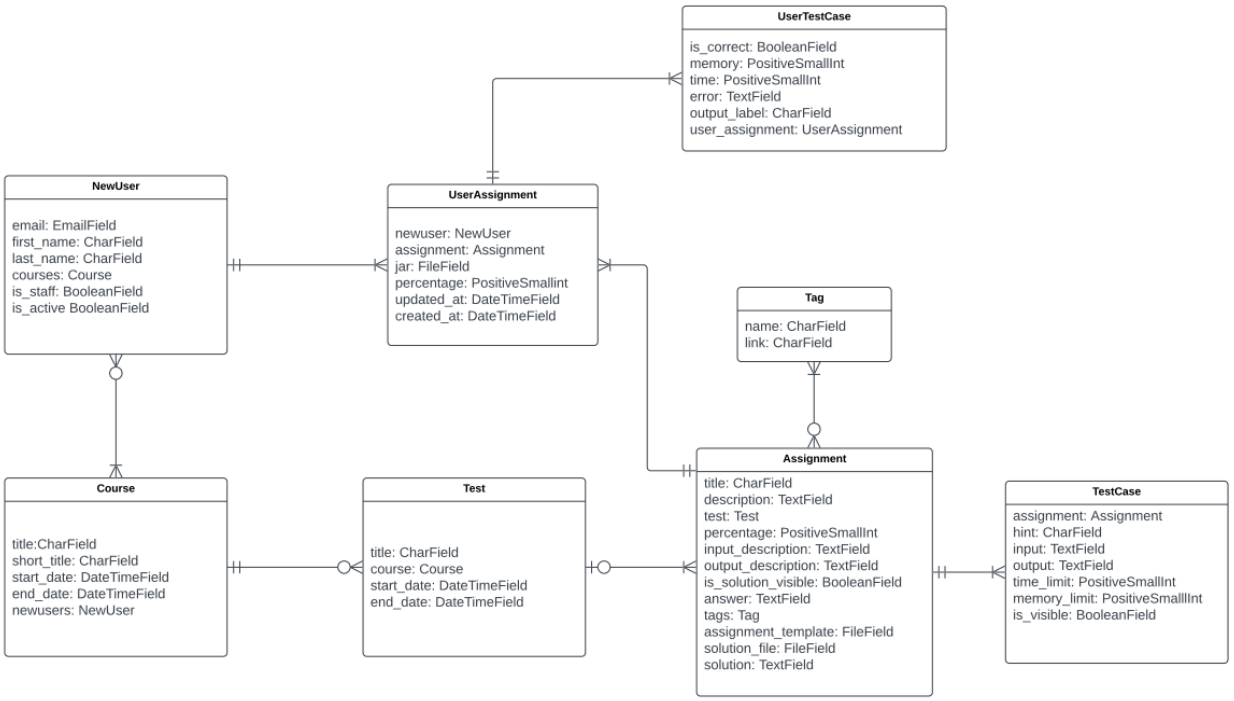
django.middleware.csrf.CsrfViewMiddleware i moguće ju je deaktivirati, no to nije preporučljivo.

4. MODELIRANJE BAZE PODATAKA

Svaki kolegij (model *Course*) ima svoj naslov, kraticu, datum početka, datum završetka i nula ili više korisnika koji su upisali taj kolegij. Kolegij se sastoji od više ispita od kojih neki mogu biti teorijski, a neki mogu u sebi sadržavati samo programske zadatke. Ispiti (model *Test*) koji sadrže programske zadatke sadrže informacije o naslovu ispita, primarni ključ kolegija na kojem se nalaze te vrijeme početka i završetka trajanja ispita. Ispit se sastoji od više zadataka (model *Assignment*) koji sadrže naslov zadatka, opis, strani ključ ispita, postotak uspješnosti, opis ulaznih podataka, opis izlaznih podataka, primjer rješenja zadatka, binarnu vrijednost koja postavlja vidljivost rješenja zadatka, relaciju na jednu ili više oznaka (model Tag), putanju do JAR datoteke koja sadrži rješenje zadatka, putanju do JAR datoteke koja sadrži predložak zadatka, rješenje zadatka i putanju do klase s JUnit testovima. Svaki zadatak sadrži više ispitnih uzoraka koji sadrže strani ključ na zadatak, savjet za rješavanje zadatka, ulaz koji se prosljeđuje u korisnički program, izlaz koji je očekivani rezultat ispravnog korisničko programa, binarnu vrijednost koja postavlja vidljivost ispitnog uzorka te zapis o memorijskom i vremenskom limitu prilikom pokretanja korisničkog programa s uzorkom. Oznake sadrže ime i hiperlink na koji se odvodi korisnika klikom na oznaku. Svaki korisnik ima korisničko ime, email, lozinku, ime i prezime. U sustav se dodatno pohranjuju informacije o aktivnosti korisnika, razini autoriteta, zadnjoj prijavi u sustav i datumu kreiranja. Korisnik i zadatak su povezani preko modela *UserAssignment* koji sadrži strani ključ korisnika i strani ključ zadatka, put do JAR datoteke koju je korisnik predao, postotak uspješnosti na zadatku te datum kreiranja i ažuriranja zadatka. Model *UserTestCase* povezuje odgovarajući ispitni slučaj s korisnikom te sadrži informacije o uspješnosti riješenog zadatka, memoriji i vremenu koje je upotrjebljeno za izvršenje ispitnog uzorka, pogrešci koja se može dogoditi prilikom izvršavanja programa s ispitnim uzorkom, oznaci izlaza programa u koju se pohranjuje vrsta iznimke nastala pogreškom ili „U redu“ u slučaju da se nije dogodila pogreška u izvršavanju, strani ključ na ispitni uzorak i strani ključ na *UserAssignment*.

Za potrebe pisanja teorijskih zadataka u aplikaciju su dodani *modeli Quiz Question, Answer, StudentQuiz* i *StudentAnswer*. Svaki kviz (model *Quiz*) predstavlja teorijski ispit te sadrži naslov, strani ključ na kolegij, opis, vrijeme početka, vrijeme završetka, korisnike koji su

dodani na ispit i ukupan broj bodova koje korisnik može ostvariti. Kvizovi se sastoje od više pitanja koji sadrže tekst pitanja i maksimalan broj bodova za točno odgovoreno pitanje. Svako pitanje može imati jedno ili više točnih odgovora. Model *StudentQuiz* sadrži informacije o uspješnosti pojedinog korisnika na teorijskom ispitu, a model *StudentAnswer* bilježi podatke o svakom odgovoru na pitanje koje se pohranjuje nakon što korisnik odgovori na pitanje. Kako bi razumijevanje povezanosti između modela bilo bolje, ER dijagram je prikazan unutar priloga A.1.



Slika 4.1 *Skraćeni prikaz ER dijagrama baze podataka*

5. AUTENTIFIKACIJA I AUTORIZACIJA

Django dolazi sa sustavom provjere autentičnosti korisnika. Upravlja korisničkim računima, grupama, dopuštenjima i korisničkim sesijama temeljenim na kolačićima. Django autentifikacijski sustav upravlja i autentifikacijom i autorizacijom. Ukratko, autentifikacija potvrđuje da je korisnik ono za što se predstavlja, a autorizacija određuje što autentificirani korisnik smije učiniti. Ovdje se izraz autentifikacija koristi za oba zadatka.

Sustav autentifikacije sastoji se od korisnika, dopuštenja koja su zapravo binarne zastavice koje označavaju smije li korisnik izvršiti određeni zadatak, grupe koje sadrže generički način primjene oznaka i dopuštenja za više od jednog korisnika, konfigurabilnog sustava za raspršivanje lozinki koji služi za prikrivanje stvarne lozinke i alata za obrasce i preglede za prijavu korisnika ili ograničavanje sadržaja.

Podrška za autentifikaciju uključena je kao Django contrib modul u *django.contrib.auth*. Prema zadanim postavkama, potrebna konfiguracija je već uključena unutar datoteke *settings.py* koja je automatski generirana prilikom stvaranja projekta. Unutar *INSTALLED_APPS* varijable se nalazi django.contrib.auth koji sadrži jezgru autentifikacijskog okvira i njegove zadane modele i *django.contrib.contenttypes* koji dopušta povezivanje dopuštenja s modelima.

U varijabli *MIDDLEWARE* bi se trebali nalaziti *SessionMiddleware* koji upravlja sesijama preko zahtjeva i *AuthenticationMiddleware* koji povezuje korisnike sa zahtjevima pomoću sesija[21].

5.1. Korisnički (User) Model

Korisnički objekti jezgra su autentifikacijskog sustava. Samo jedna klasa korisnika postoji u Djangovom okviru za provjeru autentičnosti, to jest student i administrator koriste isti korisnički objekti s posebno postavljenim atributima, a ne različite klase korisničkih objekata.

5.2. Prijava i odjava korisnika

Prijava u sustav je omogućena klikom na gumb 'Prijava' nakon kojeg je korisnik preusmjeren na stranicu prijave.

```
def login_user(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(request, username=username,
        password=password)
        if user is not None:
            login(request, user)
            return redirect('app:home')
    form = AuthenticationForm()
    return render(request, 'login.html', {'form': form})


def logout_user(request):
    logout(request)
    return redirect('app:home')
```

Na isječku koda iznad je prikazan programski kod za prijavu korisnika. Iz poslanog *POST* zahtjeva za autentifikaciju korisnika čitaju se korisničko ime i lozinka te se izvršava metoda *authenticate()* koja vrši autentifikaciju korisnika u slučaju da on postoji u bazi podataka. Ako se pošalje *GET* zahtjev, onda se napravi nova instanca forme za autentifikaciju i korisnika se preusmjerava na stranicu prijave u sustav. Za odjavu iz sustava dovoljno je pozvati *logout()* funkciju koja prima poslani zahtjev kao parametar i nakon toga preusmjeriti korisnika na početnu stranicu.

6. SUSTAV ZA UPRAVLJANJE SADRŽAJEM

6.1. Postavke unutar setting.py

Za omogućavanje integracije aplikacije sa sustavom za upravljanje sadržajem unutar liste *INSTALLED_APPS* dodane su stavke:

- 'djangocms_admin_style'
- 'django.contrib.sites'
- 'cms'
- 'menus'
- 'treebeard'
- 'sekizai'

Paket django CMS treba koristiti Djangov okvir *django.contrib.sites*. To je spojnica za pridruživanje objekata i funkcionalnosti određenim stranicama. Koristi se i zato jer je to mjesto za čuvanje naziva domena, a upotrebljava se kada jedna Django instalacija pokreće više od jedne web stranice i one se na neki način moraju razlikovati. Postavka *SITE_ID* navodi *ID* baze podataka koji sadrži Site model. Ako je postavka izostavljena, funkcija *get_current_site()* pokušat će dobiti trenutnu stranicu uspoređujući domenu s nazivom domaćina iz metode *request.get_host()*.

Moduli 'cms' i 'menus' su osnovni django CMS moduli, a 'treebeard' se koristi za upravljanje django CMS stranicom. Modul 'djangocms_admin_style' daje određeni stil koji olakšava rad s administrativnim komponentama django CMS-a. Važno je da se 'djangocms_admin_style' nalazi na listi ispred 'django.contrib.sites' jer se inače mogu desiti konflikti između modula.

CMS zahtijeva da se postavi postavka *LANGUAGES*. Unutar varijable sunavedeni svi jezici koji

se koriste u projektu. Varijabla *LANGUAGE_CODE* mora biti specificirana na neki jezik koji je naveden unutar liste *LANGUAGES*.

Za upravljanje statičkim datotekama Django koristi modul Sekizai. Uz django-sekizai mogu se rezervirati mjesta gdje se blokovi prikazuju i na različitim mjestima u predlošcima dodati ti isti blokovi. Ovo je posebno korisno za CSS i JavaScript. Predlošci mogu definirati CSS i Javascript datoteke koje će biti uključene. Pripadajući CSS će biti postavljen na vrh, a Javascript na dno. Također oznake unutar predloška ignoriraju svaki dvostruki sadržaj u jednom bloku.

Kako bi django CMS funkcionirao, potreban mu je najmanje jedan predložak za svoje stranice pa je potrebno dodati *CMS_TEMPLATES* unutar postavka. Prvi predložak na popisu *CMS_TEMPLATES* je zadani predložak projekta[22].

6.2. Postavke unutar predloška

Unutar predloška u kojem se upotrebljava sustav za upravljanje sadržajem nalazese:

- *{% load cms_tags sekizai_tags %}* koji učitava biblioteke koje unutar sebe koriste predloške koji se mogu upotrijebiti unutar predloška u kojem su specificirane.
- *{% page_attribute "page_title" %}* koji izdvaja atribut page_title stranice kako bi se mogao modificirati na različitim stranicama.
- *{% render_block "css" %}* i *{% render_block "js" %}* oznake koje dolaze iz Sekizai predloška i koje učitavaju blokove HTML-a definirane Django aplikacijama.
- *{% cms_toolbar %}* koji prikazuje django CMS alatnu traku.
- *{% placeholder "content" %}* koji definira rezervirano mjesto gdje se dodaci mogu umetnuti. Predložak treba barem jedno rezervirano mjesto za umetanje dodataka kako bi se mogao mijenjati sadržaj stranice.

7. PROGRAMSKI ZADACI

7.1. Opis programskog koda za pokretanje JUnit testova

Korisnici imaju dostupan gumb za preuzimanje predloška koji dodaje administrator. Unutar tog predloška bi se trebali nalaziti JUnit testovi koji ispituju ispravnost izvršenog zadatka.

```
junitTestsOut, junitTestErr = Popen('java -cp' +
        os.path.join(mediaPath, request.FILES['jar'].name) +';' +
        os.path.join(
                BASE_DIR, 'junit.jar junit.textui.TestRunner '
        +context['assignment'].test__class),
        stdin=PIPE, stderr=PIPE, stdout=PIPE).communicate()
context['junit'] = junitTestsOut.decode("utf-8") +
junitTestErr.decode("utf-8")
```

Isječak koda prikazuje programski kod koji je zadužen za pokretanje JUnit testova koji se nalaze unutar JAR datoteke. Klasa *Popen* je dio Pythonovog subprocess modula te omogućuje stvaranje i upravljanje procesima. *Popen* nudi veliku fleksibilnost tako da programeri mogu rješavati manje uobičajene slučajevekoji nisu obuhvaćeni konvencionalnim funkcijama. Konstruktor te klase kao prvi argument prima naredbu koju proces treba izvršiti, a prvi argument se sastojiod nekoliko dijelova odvojenih plusom koji je zapravo znak koji se koristi unutar Python programskog jezika za ulančavanje nizova[23].

Naredba *java* pokreće Java aplikaciju. To čini pokretanjem Java Virtual Machine (JVM), učitavanjem navedene klase i pozivanjem metode *main()* te klase. Glavna, *main()* metoda mora biti deklarirana kao javna i statična, ne smije vraćati nikakvu vrijednost i mora prihvatiti niz kao parametar. JVM omogućava Java programima da se izvode na bilo kojem uređaju ili operativnom sustavu i optimizira programsku memoriju[24].

Prema zadanim postavkama, prvi argument koji nije opcija *java* naredbe je ime klase koju treba pozvati. Ako je naveden *-jar*, tada je njegov argument naziv JAR datoteke koja sadrži

klase i sve ovisnosti potrebne za izvršavanje aplikacije. Argumenti nakon naziva Java *class* datoteke ili naziva JAR datoteke prosljeđuju se metodi *main()*.

Java *class* datoteka je datoteka koja sadrži Java bajt kod i ima ekstenziju *.class* koju može izvršiti JVM. Java *class* datoteku stvara Java kompilator iz *.java* datoteka. Java datoteka ima nastavak .java i može sadržavati jednu ili više klasa. Dakle, ako .java datoteka ima više od jedne klase, tada će se svaka klasa kompilirati u zasebne Java *class* datoteke[25]. U ovom primjeru korištena je naredba java sa zastavicom *-cp* koja omogućuje unos popisa direktorija, JAR arhiva i ZIP arhiva odvojenih točkom i zarezom (;) unutar kojih se nalaze Java *class* datoteke.

Kako bi se naredba mogla pokrenuti na svim operativnim sistemima koristi se Pythonov OS modul. OS modul dolazi pod standardnim modulima Pythona i u Pythonu pruža funkcije za interakciju s operativnim sustavom te prijenosni način korištenja funkcionalnosti ovisne o operativnom sustavu. Modul *os.path* je podmodul OS modula u Pythonu koji se koristi za manipulaciju naziva putanje. Metoda *os.path.join()* u Pythonu inteligentno spaja jednu ili više komponenti putanje. Ova metoda spaja različite putanje s točno jednim razdjelnikom direktorija iza svakog dijela putanje osim posljednje komponente. Ako je posljednja komponenta putanje koja se spaja prazna, tada se na kraju stavlja razdjelnik direktorija. Ako komponenta staze predstavlja apsolutnu stazu, tada se sve prethodno spojene komponente odbacuju i spajanje se nastavlja od komponente apsolutne staze[26].

Varijabla *mediaPath* je inicijalizirana na vrhu datoteke i sadrži apsolutnu putanju do JAR datoteka koje su predane od strane korisnika aplikacije tijekom izvršenja zadataka.

Sve korisničke datoteke koje su učitane u aplikaciju spremaju se u direktorij pod nazivom *media* koji se nalazi u polazišnom direktoriju, a putanja je definirana programskim kodom prikazanim u sljedećem isječku koda.

```
mediaPath = os.path.join(BASE_DIR, 'media')
```

Putanja do polaznog direktorija je spremljena u varijablu *BASE_DIR*. Unutar varijable *request.FILES* nalaze se svi podaci o datoteci koja je učitana trenutnim zahtjevom. Unutar *FILES*

će se nalaziti podaci samo ako je metoda zahtjeva bila *POST* i ako je forma koja je poslana na zahtjev imala *enctype="multipart/formdata"*. U suprotnom, *FILES* će biti prazan objekt.

Forma za predaju zadatka sadrži polje pod nazivom *jar* za predaju korisničkog zadatka pa se preko *request.FILES['jar'].nam*e dohvaća puno ime predane datoteke. Sve datoteke potrebne za izvršavanje testova nalaze se unutar *junit.jar* datoteke koja se nalazi unutar glavnog direktorija. Unutar *junit.jar* datoteke nalazi se i *TestRunner* klasa koja na ulazu prima klasu unutar koje se nalaze testovi, a na izlazu daje rezultat JUnit testova. Administrator mora dodati putanju do svake testne klase kako bi se testovi uspješno izvršili. U slučaju dase klasa s testovima nalazi unutar paketa potrebno je specificirati ime paketa i koristiti točku kao separator između imena paketa i klase.

```
package GRUPA_A;

import static org.junit.Assert.*;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.JUnit4;
import junit.framework.TestCase;


@RunWith(JUnit4.class)
public class PolicijaTest extends TestCase{


        @Test
        public void testSustav1() {

                Nacelnik n = new Nacelnik ("Dean Nacelnik");Po
                licajac p = n.dodajPol("Clancy Wiggum");
                assertEquals("Clancy Wiggum", p.getIme());

        }
}
```

Bitno je napomenuti kako svaki test unutar izvršne testne klase mora sadržavati dekorator *@Test* iznad funkcije i ime funkcije mora započeti izrazom „test“. Nadalje klasa mora naslijediti klasu *TestCase* i iznad klase je potrebno dodati dekorator *@RunWith(JUnit4.class)* kao što je prikazano na isječku koda.

Parametri *stdin*, *stdout* i *stderr* specificiraju standardni unos, standardni izlaz i grešku pri izvršavanju. Valjane vrijednosti su *PIPE*, *DEVNULL*, postojeći deskriptor datoteke (pozitivan cijeli broj), postojeći objekt datoteke s važećim deskriptorom datoteke i *None*. *PIPE* označava da treba stvoriti novu cijev za dijete procesa. *DEVNULL* označava da će se koristiti posebna datoteka *os.devnull*. Uz zadane postavke *None*, neće se dogoditi nikakvo preusmjeravanje; djetetove datoteke će se naslijediti od roditelja. Kako bi dobili izlaz i grešku potrebno je postaviti parametre na *PIPE* [27].

Nad instancom klase *Popen* pozvana je metoda *communicate()* koja izvršava naredbu te na izlazu vraća dvije vrijednosti od kojih je prva vrijednost rezultat izvođenja procesa, a druga vrijednost sadrži informacije o pogrešci pri izvršavanju. Samo jedna od tih dviju varijabli će sadržati neku vrijednost, to jest može doći samo do uspješnog ili samo do neuspješnog izvršavanja procesa.

Budući da su dobivene vrijednosti u *utf-8* formatu potrebno ih je dekodirati. Dekodiranje se obavlja tako da se nad varijablom koju želimo dekodirati pozove metoda *decode()* koja kao argument prima vrstu kodiranja koju želimo dekodirati. Izlaz i pogreška procesa se spremaju u *context* objekt koji se prosljeđuje u html predložak te se korisniku prikazuje rezultat izvršenja testova u obliku padajućeg izbornika koji je izveden uz pomoć Bootstrapa.

7.2. Automatizirana provjera zadataka

Provjera ispravnosti riješenih zadataka se ne treba odvijati isključivo JUnit testovima. Administrator može zadati određeni broj ispitnih slučajeva te na temelju ulaznih i izlaznih varijabli provjeriti ispravnost predanog zadatka.
Pokretanje zadataka se pokreće *check_output()* funkcijom koja ima slična svojstva kao i

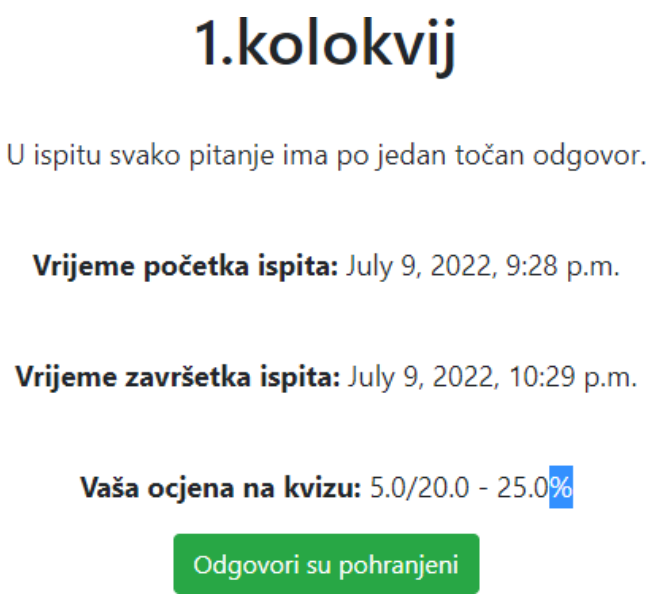
konstruktor klase *Popen*. Ponovo se koristi naredba *java*, no, ovaj put sa zastavicom *-jar*. Zastavica omogućuje izvršavanje programa enkapsuliranog u JAR datoteku. Argument nakon zastavice je naziv JAR datoteke. Kada se koristi *-jar*, navedena JAR datoteka je izvor svih korisničkih klasa, a ostale postavke staze klase se zanemaruju. Prilikom kreiranja JAR datoteke bitno je specificirati put do main funkcije. Unutar Eclipse uređivača to se može postići tako da se prilikom kreiranja JAR datoteke pritisne gumb „Next“ dok se ne dođe do zadnjeg okvira i pritom se klikom na „Browse“ protvara dijaloški okvir sa svim main funkcijama unutar projekta. Potrebno je odabrati željenu funkciju i onda kliknuti na gumb „Finish“.

U argument *input* postavljamo ulaz u JAR datoteku u *utf-8* formatu. Vrijednost ulaza je spremljena unutar *TestCase* objekta pa se ulaz u *utf-8* formatu dobije pozivanjem metode *encode()*. Svaki ispitni uzorak ima i vremensko ograničenje unutar kojeg se mora izvršiti, a trajanje procesa unutar *check_output()* funkcije se specificira argumentom *timeout* koji je postavljen na vremenski limit ispitnog uzorka. Kako bi korisnici imali informaciju o trajanju izvršenja zadatka za odgovarajući ulaz potrebno je izmjeriti vrijeme prije i nakon izvršenja zadatkai potom oduzeti vrijednosti kako bi dobili konačnu vrijednost u milisekundama.

8. TEORIJSKI ZADACI

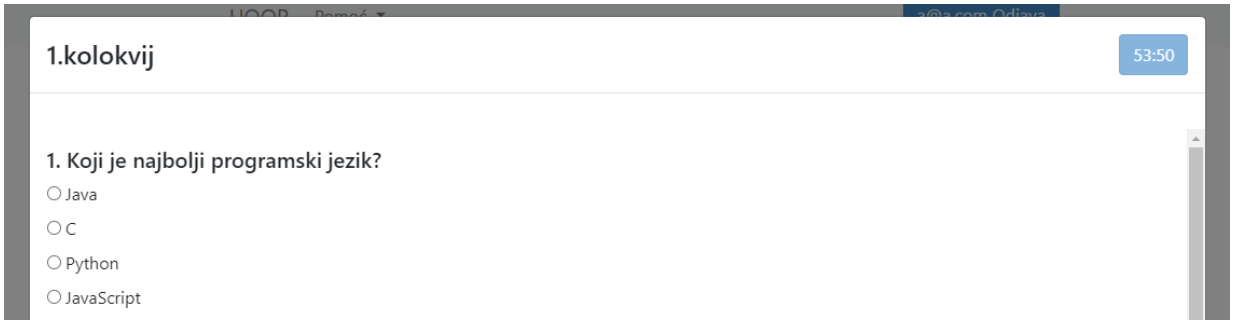
Budući da teorijski zadaci sadrže samo pitanja i odgovore, modeli se razlikuju od modela koji se koriste za pisanje programskih zadataka i povodom toga za teorijske zadatke su napisani pogledi odvojeni od programskih zadataka kako bi održavanje sustava postalo praktičnije.

Administratoru je omogućeno određivanje vrijeme početka ispita te vrijeme završetka ispita. Oduzimanjem završetka od početka dobivamo trajanje ispita. Putem administratorskog sučelja može se modificirati i ukupan broj bodova te broj bodova na svakom pitanju pojedinačno. Ukupan broj bodova ostvarenih na ispitu je spremljen u tablicu *app_studentquiz* koja sadrži relaciju na studenta i na teorijski ispit koji je predao. Kada se taj zapis kreira, student više ne može pristupiti ispitu. Jedini način da student ponovno pristupi istom ispitu je da se taj zapis obriše.



Slika 8.1 *Informacije prikazane studentu nakon predaje teorijskogzadatka*

Nakon što korisnik iz liste ispita odabere ispit kojemu želi pristupiti on je preusmjeren na rutu koja dohvaća opis i trajanje ispita i studentu se prikazuje gumb kojim može pristupiti ispitu. Stranica koja je prikazana korisniku nalazi se na slici 8.1.



Slika 8.2 *Dijaloški okvir sa jednim primjerom pitanja*

Klikom na gumb „Pokreni ispit“ otvara se okvir sa slike 8.2. Prilikom predaje zadatka ili isteka vremena studentu je odmah dostupna informacija o ostvarenim bodovima. U lijevom kutu je prikazano ime ispita, a u desnom preostalo vrijeme pitanja. Prilikom gubitka veze ili ponovnog pokretanja stranice student može ponovo pristupiti ispitu uz uvjet da ispit još uvijek nije završio.

9. ADMINISTRATORSKO SUČELJE

Jedan od dijelova Djanga je automatsko administratorsko sučelje. Administratorsko sučelje čita metapodatke iz modela kako bi korisnici mogli upravljati sadržajem na web stranici i ima mnogo opcija za prilagodbu.

9.1. Prilagodba sučelja

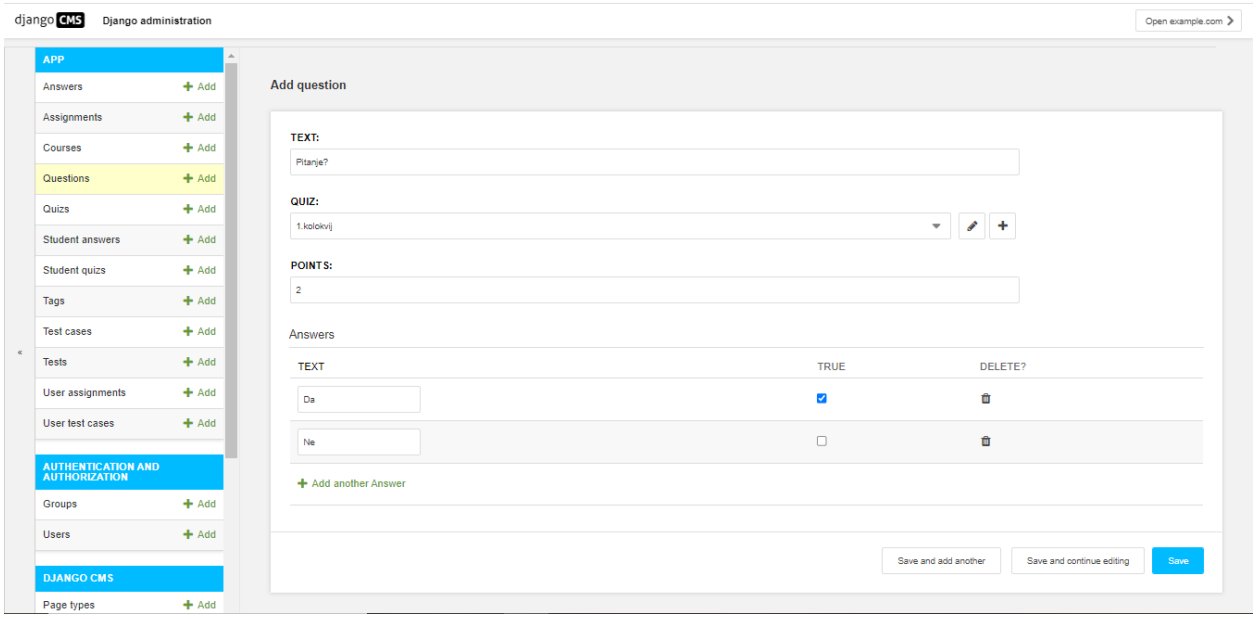
Iako je sučelje generirano na temelju tablice i modela iz baze podataka potrebno je podesiti sučelje za uporabu i prilagoditi ga u nekim situacijama.

```
admin.site.register(StudentQuiz)

class StudentAnswerInline(admin.TabularInline):
    model = StudentAnswer
    extra = 0

class StudentQuizAdmin(admin.ModelAdmin):
    inlines = [StudentAnswerInline,]

admin.site.unregister(StudentQuiz)
admin.site.register(StudentQuiz, StudentQuizAdmin)
```

U kratkom isječku koda prikazano je dodavanje modela i jednostavno modificiranje prikaza stranice na administratorskom sučelju. Kako bi se prikaz na administratorskom sučelju prilagodio dodana je klasa *AnswerInline* koja nasljeđuje klasu *TabularInline.* Klasa *TabularInline* posjeduje atribut *model* unutar kojeg se postavlja model koji će se nalaziti na istoj stranici ispod roditelja tog modela. Sa parametrom *extra* se definira broj prikaza praznih polja za unos tog modela. Model se registrira metodom *admin.site.register()* koja kao prvi argument prima model kojeg želimo prikazati na administratorskom sučelju, a kao drugi argument prima klasu *ModelAdmin* koje reprezentira model unutar administratorskog sučelja. Klasa *StudentQuizAdmin* nasljeđuje klasu *ModelAdmin* te unutar atributa *inlines* specificira polje modela koji će se prikazati u retcima ispod glavnog modela.

9.1. Dodavanje i modificiranje provjera znanja

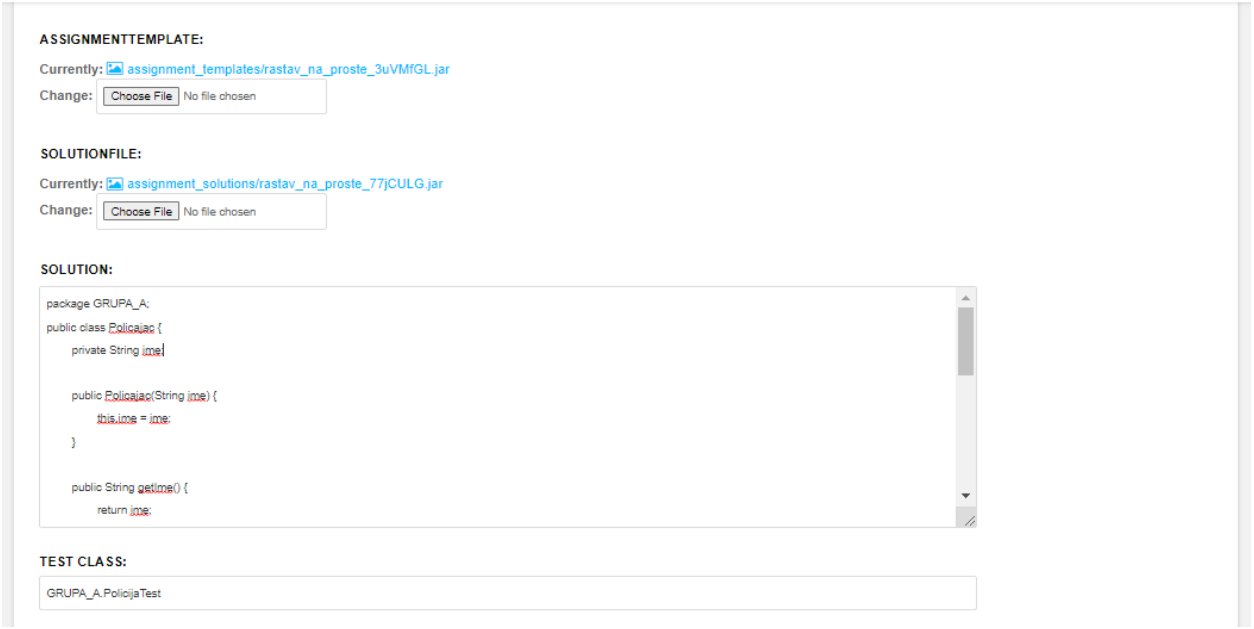
Klikom na *Quizs* prikazuje se lista svih teorijskih provjera znanja koje se nalaze unutar baze podataka. Klikom na gumb *Add+* dodaje se novi zapis. Na provjeri je moguće dodati relaciju na kolegij kojem pripada čime će ona biti dodana u listu unutar kolegija i automatski vidljiva studentima. Ako se relacija na kolegij ne specificira onda provjera neće biti prikazana. Unutar polja students dodaju se korisnici kojima je odobren pristup provjeri. Budući da je ponekad potrebno postaviti trajanje provjere na više od jednog dana (na primjer probne provjere) vrijeme početka i završetka su zapisani kao kombinacija datuma i vremena.



Slika 9.1 *Uređivanje pitanja*

Lista svih pitanja se prikazuje klikom na gumb *Questions* unutar *APP* izbornika. Na pitanje se može dodati broj bodova koje student može ostvariti prilikom odabira točnog odgovora. Na dnu glavne forme se nalaze mogući odgovori na pitanja. Točna pitanja se dodaju označavanjem kvačice ispod sekcije *True*. Svako pitanje se može dodati provjeri odabirom provjere sa padajućeg izbornika unutar sekcije *QUIZ*. Nakon isteka vremena ili predaje provjere znanja u bazi se kreira zapis na kojem su spremljene informacije o ostvarenim bodovima na provjeri. Lista na kojoj su prikazana sva riješenja dostupna je odabirom *Student quizs* unutar *App* izbornika.

Slično kao i kod teorijske provjere, za dodavanje provjere sa programskim zadacima zadacima potrebno je unijeti naslov i trajanje, a opcionalno i relaciju na kolegij unutar kojeg želimo da se provjera nalazi. Na zadacima se, nakon kreiranja provjere, može dodati relacija na provjeru kako bi zadaci bili vidljivi unutar provjere. Kako bi studentima bilo jasnije o kojoj vrsti zadatka se radi, administrator može dodati oznake (*Tags*) na svaki zadatak.

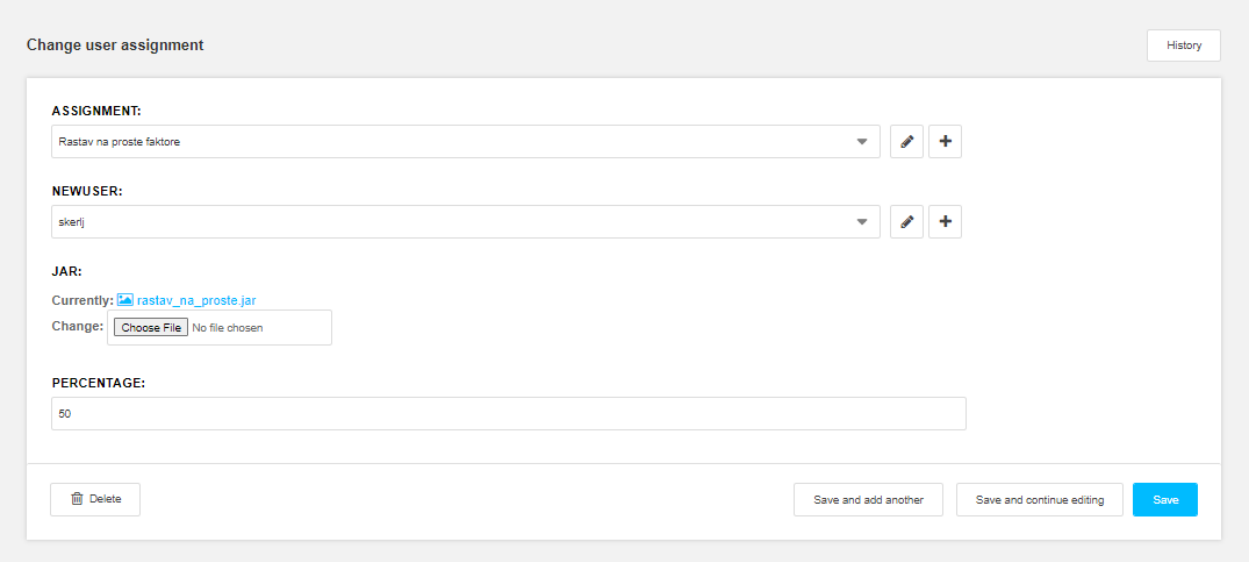


Slika 9.2 *Polja koje administrator može modificirati na programskom zadatku*

Administratorima je omogućeno dodavanje predložaka za riješavanje zadatka te prilaganje riješenja u obliku teksta ili JAR datoteke. U slučaju da se u predlošku za riješavanje zadatka nalazi datoteka unutar koje se nalaze JUnit testovi, administrator mora specificirati put do te datoteke unutar rubrike kako bi se testovi uspješno pokrenuli. Put se sastoji od imena paketa i klase koja se nalazi unutar tog paketa. Za separiranje imena paketa od imena klase potrebno je koristiti točku kao separator bez razmaka. Prema primjeru sa slike ime paketa je *GRUPA_A*, a ime klase unutar koje se nalaze testovi je *PolicijaTest*.

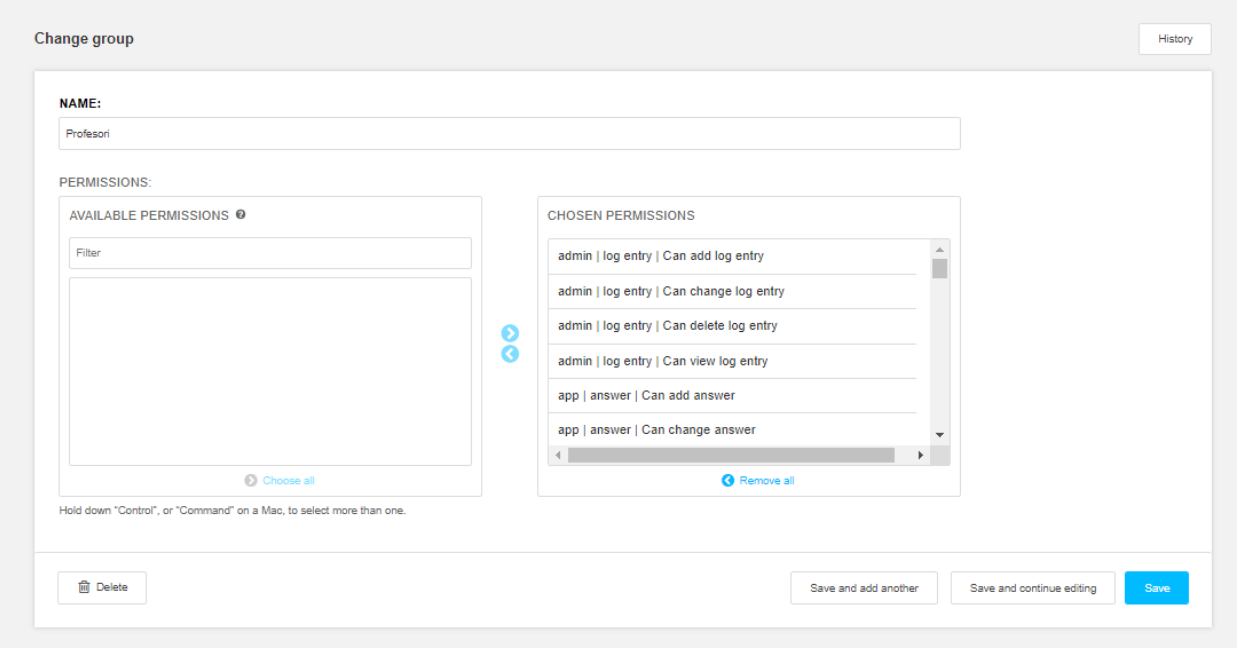
Zadaci mogu imati i određeni broj ispitnih uzoraka koji provjeravaju izvršeni kod na temelju izlaza i ulaza. Za kreiranje ispitnog uzorka administrator mora posjedovati riješenje zadatka.

Najbolji način za kreiranje ispitnih uzoraka je putem ljuske. U ljusku je potrebno upisati *java -jar* nakon čega slijedi apsolutni ili relativni put do datoteke unutar koje se nalaze riješenja i razmakom odvojen ulaz. Ulaz je potrebno zaljepiti u polje *Input*, a izlaz u polje *Output* i pritom je potrebno paziti na razmake i nove linije.



Slika 9.3 *Model UserAssignment unutar administratorskog sučelja*

Kada student prvi put preda zadatak kreira se zapis *UserAssignment* unutar kojeg se nalazi JAR datoteka koju je student predao te postotak uspješnosti na tom zadatku. Kako bi se izbjeglo pretrpavanje baze podataka sa riješenima, svakom novom predajom riješenja za isti zadatak ažurira se postojeći zapis koji na sebi sadrži relaciju na studenta i provjeru znanja koju trenutno riješava. Unutar sučelja omogućena je i detaljna provjera svakog ispitnog uzorka unutar liste *User test cases*. Na modelu je zapisan izlaz programa kojeg je student predao te zapis o pogrešci prilikom izvršavanja u slučaju da je došlo do greške.



Slika 9.4 *Stranica za dodavanje grupa unutar administratorskog sučelja*

Administratorsko sučelje omogućuje dodavanje novih grupa. Lista svih grupa dostupna je klikom na *Groups* unutar *App* izbornika. Svaka grupa sadrži ime i dopuštenja koja grupa ima. Dopuštenja su sastavljena od vrste, modela i opisa. Na primjer dopuštenje *'app | answer | Can add answer'* govori da grupa sa tim dopuštenjem može dodavati odgovore na pitanja unutar aplikacije.

Unutar izbornika *Users* je prikazana lista svih korisnika. Osnovne informacije koje korisnik posjeduje su korisničko ime, ime, prezime, email i lozinka koja je enkriptirana. Korisnik može sadržati grupu i time nasljeđuje sva prava koja su dodijeljena toj grupi. Svakom korisniku je moguće dodijeliti i zasebna prava. Postavljanjem kvačice unutar polja *Superuser status* korisniku je omogućen login na administratorsko sučelje te posjeduje maksimalnu razinu prava. Umjesto da se korisnici obrišu moguće je maknuti kvačicu sa polja *Active* čime je korisnik prividno obrisan. Korisnik se neće moći ulogirati, ali zapis će postojati u bazi podataka. Unutar sučelja se nalazi datum i vrijeme posljednje prijave korisnika u sustavi i datum kreiranja korisnika s administratorskog sučelja.

10. UPUTSTVA ZA POKRETANJE APLIKACIJE

10.1. Kloniranje i pokretanje projekta

Projekt je smješten na Githubu i otvoren je javnosti. Za instalaciju projekta potrebno je preuzeti git i Python s verzijom naznačenom u tablici 2.1. Nakontoga potrebno je instalirati standardni upravitelj paketa za Python naredbom *python -m pip install –upgrade pip*.

Pip omogućuje instaliranje i upravljanje paketima koji nisu dio Python standardne biblioteke[20]. Nakon toga potrebno je ući u željeni direktorij unutar kojeg će se nalaziti repozitorij s izvornim kodom aplikacije. S ljuske je potrebno pokrenuti naredbu *git clone https://github.com/LukaSkrlj/uoop-web-app.git* kako bi se repozitorij preuzeo lokalno. Nakon što se repozitorij preuzme potrebno je pokrenuti *py -3 -m venv .venv* kako bi se inicijalizirao *.venv* direktorij u kojem će se nalaziti svi paketi potrebni za pokretanje projekta. Pokretanjem *.venv\Scripts\activate* aktivira se virtualno okruženje unutar kojeg je potrebno instalirati sve pakete unosom naredbe *pip install -r requirements.txt*.

Nakon instalacije svih paketa potrebno je podesiti bazu. Sustav za upravljanje bazama podataka koji se koristi unutar projekta je MySql. Unutar direktorija u kojem se nalazi projekt potrebno je stvoriti *.env* datoteku unutar koje trebaju biti specificirane varijable *DB_NAME* koje predstavlja ime baze podataka, *DB_USER* koje predstavlja korisničko ime, *DB_HOST* koji predstavlja ip adresu MySql servera, *DB_PORT* koji predstavlja *port* servera i *DB_PASSWORD* koji predstavlja lozinku baze podataka.

Nakon kreiranja *.env* datoteke potrebno je migrirati bazu naredbom *python manage.py migrate* te pokrenuti naredbu *python manage.py runserver <broj_porta>* gdje *<broj_porta>* označava port na kojem će se pokrenuti web aplikacija. Potrebno je obratiti pozornost da *DB_PORT* i broj *porta* na kojem se pokreće web aplikacija ne budu isti. Najbolja praksa je postaviti *DB_PORT* na vrijednost 3306 i pokrenuti naredbu *python manage.py runserver* koja pokreće aplikaciju na portu 8000. Kako unutar baze nije kreiran korisnik potrebno ga je kreirati pokretanjem naredbe *python manage.py createsuperuser*. Ova je naredba dostupna samo ako je instaliran Djangov

sustav provjere autentičnosti (*django.contrib.auth*).

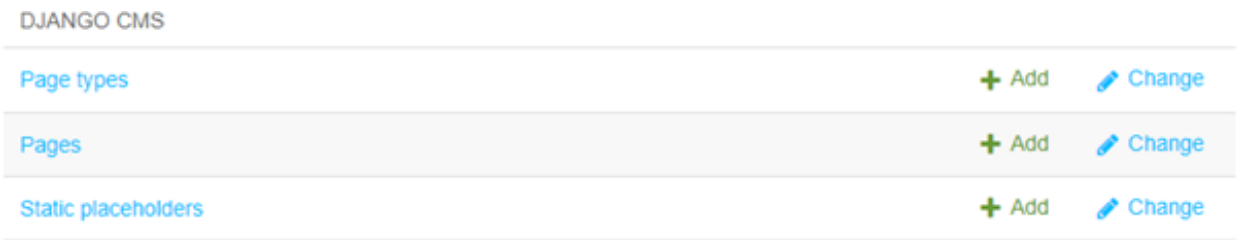
Prilikom pokretanja naredbe stvara se račun super korisnika (korisnik koji ima sva dopuštenja). Naredba će tražiti lozinku za novi račun super korisnika. Kada se pokreće neinteraktivno, lozinka se može postaviti postavljanjem varijable *DJANGO_ SUPERUSER_ PASSWORD*. U protivnom neće biti postavljena lozinka, a račun superkorisnika neće se moći prijaviti dok se lozinka za njega ne postavi ručno.

Nakon što je dovršeno gore opisano postavljanje, može se upotrijebiti ugrađenu naredbu *python manage.py cms check*. Naredba potječe iz django CMS-a, a služi za identificiranje i instaliranje drugih komponenata, provjerava konfiguraciju, instalirane aplikacije i bazu podataka te izvještava korisnika o svim problemima.

Administrator može dodavati nove korisnike i upravljati bazom podataka preko sučelja koje se nalazi na *http: // 127. 0. 0. 1: 8000/ admin* .

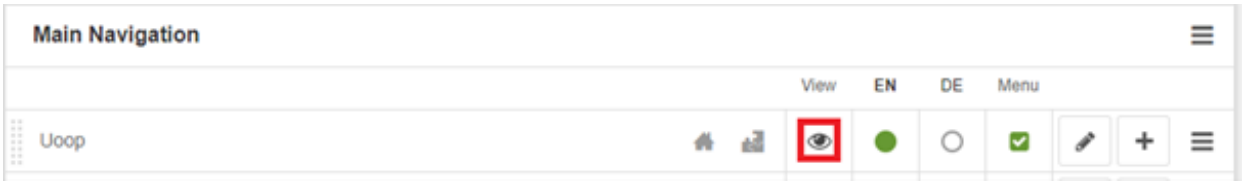
10.2. Integracija u sustav za upravljanje sadržajem

Pokretanjem aplikacije dostupno je samo administratorsko sučelje na kojem je potrebno podesiti postavke sustava.



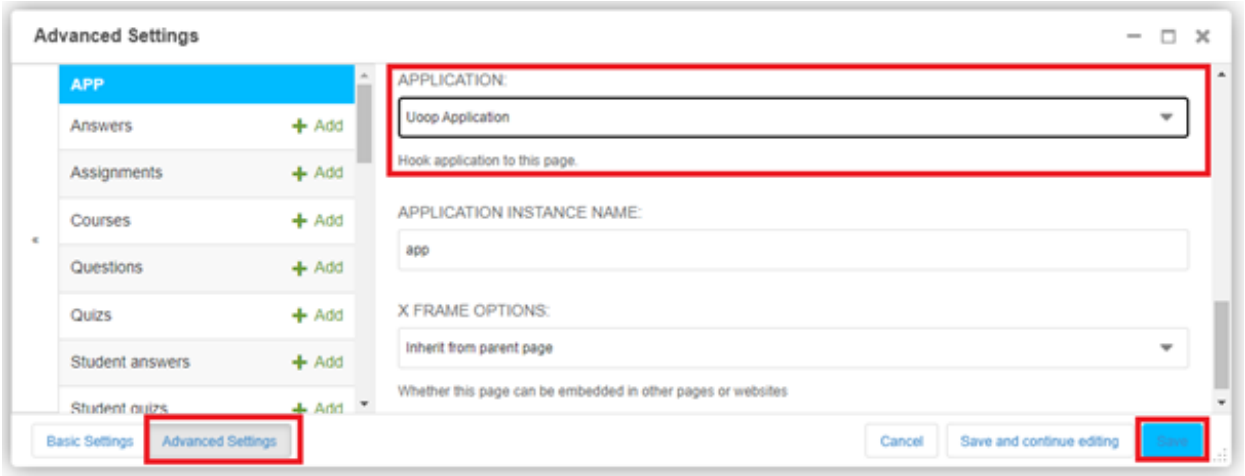
Slika 10.1 *Rubrika DJANGO CMS unutar administratorskog sučelja*

Otvaranjem administratorskog sučelja prikazat će se lista svih entiteta u bazi koje je moguće modificirati. Za dovršiti integraciju aplikacije u CMS potrebno je dodati novu stranicu koja se dodaje unutar sekcije „DJANGO CMS“ koja je prikazana na slici 10.1. Klikom na gumb „Add“ otvara se forma unutar koje je potrebno upisati osnovne podatke o stranici. Obavezna polja su samo naslov stranice i „slug“ koji predstavlja dio naslova koji se koristi u URL-u.



Slika 10.2 *Lista svih stranica ununtar administratorskog sučelja*

Nakon što su promjene spremljene, administratora se preusmjerava na prikaz liste svih stranica gdje je potrebno kliknuti na „view“ gumb sa slike 10.2 kako bi se otvorila kreirana stranica. Na ekranu će se prikazati prazna stranica s alatnom trakom na kojoj je potrebno kliknuti gumb „Pages“ ispod kojeg se prikaže padajući izbornik u kojem je potrebno odabrati „Advanced settings“. Prilikom odabiranja gumba otvorit će se dijaloški okvir koji pri dnu sadrži sekciju „Application“ unutar koje je potrebno odabrati „Uoop Application“. Spremanjem promjena zatvara se prozor i administratora se preusmjerava na početnu stranicu aplikacije. Ovime završavaintegracija i aplikacija je spremna za korištenje.



Slika 10.3 *Napredne postavke na odabranoj stranici*

11. ZAKLJUČAK

Ponekad je moguće automatizirati procese koji naizgled nisu najintuitivniji za automatizaciju. Automatizacija bi pritom trebala služiti za olakšavanje rada korisnika sustava pa je potrebno pobrinuti se za sve rubne slučajeve i za što bolje korisničko iskustvo.

Za kreirati aplikaciju koja provjerava rezultate programskih kodova potreban je širok spektar znanja i sposobnost baratanja raznovrsnim tehnologija u računalnom inženjerstvu. Potrebno je i osigurati zaštitu protiv zlouporabe sustava i neautoriziranog pristupa.

Studentima je aplikacijom omogućeno samostalno vježbanje zadataka kod kuće, dok je profesorima dan dobar uvid ostvarenog napretka pojedinog studenta, omogućeno im je pokretanje ispita kao i sustav za upravljanje sadržajem.

Sustav za upravljanje sadržajem ima neke prednosti i mane. Iako sustav omogućuje totalnu manipulaciju nad sadržajem unutar aplikacije, korisnik sustava treba biti upoznat kako ga koristiti, a integracija sustava nije toliko jednostavna za početnike jer je dokumentacija poprilično opsežna.

BIBLIOGRAFIJA

[1] Python Software Foundation: “What is Python? Executive Summary“, s Interneta, https://www.python.org/doc/essays/blurb/, 20. kolovoza 2022.

[2] Python Software Foundation: "Applications for Python", s Interneta, https://www.python.org/about/apps/#web-and-internet-development, 20. kolovoza 2022.

[3] Reza G.: "Django introduction", s Interneta, https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction, 20. kolovoza 2022.

[4] Django CMS association: "Lean enterprise content management", s Interneta, https://www.django-cms.org/en/, 20. kolovoza 2022.

[5] Django CMS association: "Django CMS documentation", s Interneta, https://docs.django-cms.org/en/latest/, 20. kolovoza 2022.

[6] Tutorialspoint: "Junit Overview", s Interneta, https://www.tutorialspoint.com/junit/junit_overview.htm, 20. kolovoza 2022.

[7] Jordana A.: "What Is Bootstrap?", s Interneta, https://www.hostinger.com/tutorials/what-is-bootstrap, 20. kolovoza 2022.

[8] Python Software Foundation: "Virtual Environments and Packages", s Interneta, https://docs.python.org/3/tutorial/venv.html, 20. kolovoza 2022.

[9] Python Software Foundation: "Django-environ", s Interneta, https://pypi.org/project/python-environ/, 20. kolovoza 2022.

[10] The Daring Fireball Company LLC: "Markdown", s Interneta, https://daringfireball.net/projects/markdown/, 20. kolovoza 2022.

[11] The Python-Markdown Project: "Python-Markdown", s Interneta, https://python-markdown.github.io/, 20. kolovoza 2022.

[12] Microsoft: "Learn to code with Visual Studio Code", s Interneta, https://code.visualstudio.com/learn, 20. kolovoza 2022.

[13] Oracle: "MySQL Workbench", s Interneta, https://www.mysql.com/products/workbench/, 20. kolovoza 2022.

[14] Scott C.: "Git", s Interneta, https://git-scm.com/, 20. kolovoza 2022.

[15] Tutorialspoint: "Eclipse - Overview", s Interneta, https://www.tutorialspoint.com/eclipse/eclipse_overview.htm, 20. kolovoza 2022.

[16] The Appory, Inc.: "Understanding Django Views", s Interneta, https://ultimatedjango.com/learn-django/lessons/understanding-django-views/, 20. kolovoza 2022.

[17] Django Software Foundation: "The Django template language", https://docs.djangoproject.com/en/4.0/ref/templates/language/, 20. kolovoza 2022.

[18] Django Software Foundation: "Models", https://docs.djangoproject.com/en/4.0/topics/db/models/, 20. kolovoza 2022.

[19] Django Software Foundation: "Creating forms from models", https://docs.djangoproject.com/en/4.0/topics/forms/modelforms/#modelform, 20. kolovoza 2022.

[20] Real Python: "Using Python's pip to Manage Your Projects' Dependencies", https://realpython.com/what-is-pip/, 20. kolovoza 2022.

[21] Django Software Foundation: "User authentication in Django", https://docs.djangoproject.com/en/4.0/topics/auth/, 20. kolovoza 2022.

[22] Django CMS association: "How to install django CMS by hand", https://docs.django-cms.org/en/latest/how_to/install.html, 20. kolovoza 2022.

[23] Oracle: "The java Command", https://docs.oracle.com/en/java/javase/13/docs/specs/man/java.html, 20.kolovoza 2022.

[24] IDG Communications, Inc.: "What is the JVM? Introducing the Java Virtual Machine", https://www.infoworld.com/article/3272244/ what-is-the-jvm-introducing-the-java-virtual-machine.html, 20. kolovoza 2022.

[25] Anshul A.: "Java Class File", https://www.geeksforgeeks.org/ java-class-file/., 20. kolovoza 2022.

[26] GeeksforGeeks: "Python | os.path.join() method", https://www. geeksforgeeks.org/python-os-path-join-method/, 20. kolovoza 2022.

[27] Python Software Foundation: "Subprocess management", https://docs. python.org/3/library/subprocess.html#subprocess.check_output, 20. kolovoza 2022.

POJMOVNIK

API Application Programming Interface.

CMS Content Management System-a.

CSRF Cross-Site Request Forgery.

CSS Cascading Style Sheets.

FTP File Transfer Protocol.

HTML Hypertext Markup Language.

HTTP Hypertext Transfer Protocol.

IMAP Internet Message Access Protocol.

JAR Java ARchiven.

DRY Don't Repeat Yourself.

CRUD Create Read Update Delete.

MVT Create Read Update Delete.

JSON JavaScript Object Notation.

JVM Java Virtual Machine.

RSS Really Simple Syndication.

SQL Structured Query Language.

SSH2 Secure Shell 2.

URL Uniform Resource Locator.

XML extensible markup language.

SAŽETAK

Kroz ovaj rad je opisana aplikacija koja je namijenjena automatiziranom pisanju ispita na kolegiju Uvod u objektno orijentirano programiranje, to jest za provjeru ispravnosti zadataka napisanih u programskom jeziku Java. Aplikacija je izgrađena na softverskom okviru Django koji je baziran na programskom jeziku Python. U prvim poglavljima su ukratko opisane tehnologije koje su korištene za uporabu projekta zajedno s nekim značajkama koje su pridonijele odabiru tih tehnologija. Nakon toga slijedi opis procesa koje Django izvršava kada se na poslužitelj pošalje neki zahtjev. Potom je opisan postupak modeliranja baze podataka, postupak pokretanja aplikacije, Djangov sustav autentikacije i autorizacije kao i način na koji funkcionira CMS. Projekt sadrži slike iz aplikacije i slike s isječcima koda koji su pisani unutar Visual Studio Code i Eclipse uređivača. Dodatna dokumentacija i cijeli izvorni kod javno je dostupan i nalazi se na https://github.com/LukaSkrlj/uoop-web-app.

***Ključne riječi* — Django, Python, CMS, Uvod u objektno orijentirano programiranje**

Abstract

This thesis describes an application intended for automated writing of exams in the course Introduction to object-oriented programming, that is, for checking the correctness of tasks written in the Java programming language. The application is built on the Django software framework, which is based on the Python programming language. In the first chapters, the technologies that were used for creating this project are briefly described, along with some features that contributed to the selection of these technologies. This is followed by a description of the processes that Django executes when a request is sent to the server. Then the database modeling procedure, the application startup procedure, Django's authentication and authorization system, as well as the way CMS works are described. The project contains images from the application and images with code snippets written within Visual Studio Code and the Eclipse editor. Additional documentation and complete source code are publicly available at https://github.com/LukaSkrlj/uoop-web-app.

***Keywords* — Django, Python, CMS, Introduction to object-oriented programming**