Univerzitet u Beogradu

MATEMATIČKI FAKULTET

Master rad

Tema: KORIŠĆENJE PROGRAMSKOG JEZIKA PYTHON U IZRADI ŠKOLSKIH ADMINISTRATIVNIH APLIKACIJA

Profesor:
dr Dušan Tošić

Student:
Jovana Radotić

Beograd, septembar 2013

Apstrakt:

Python je programski jezik visokog nivoa, opšte namene, koji podržava različite programske paradigme: proceduralnu, funkcionalnu i objektno-orijentisanu. Njegova veoma čista i čitljiva sintaksa, zajedno sa interpreterskom prirodom, čini ga idealnim jezikom za razvoj različitih aplikacija. Django je jedno od najpopularnijih razvojnih okruženja za razvoj Web aplikacija koje je pisano na programskom jeziku Python. Omogućava razvoj složenih Web aplikacija, uz manje uloženog vremena i truda. Koristi MVC obrazac za arhitekturu softvera, koji razdvaja aplikaciju na tri dela: Model, View i Controller.

U ovom radu opisan je razvoj Web aplikacije korišćenjem Django razvojnog okruženja. Ova aplikacija putem Interneta olakšava komunikaciju između nastavnika i roditelja, i omogućava roditeljima uvid u ocene i izostanke svog deteta u školi.

Abstract:

Python is a general-purpose, high-level programming language that supports various programming styles: imperative, functional, and object-oriented. It's very clean and readable syntax, together with its interpreted nature, makes it an ideal language for developing a wide variety of applications. Django is one of the the most popular web application frameworks written in Python. It enables you to build complex web applications with less time and effort. It follows the MVC design pattern that separates three parts of a web-application: Model, View i Controller.

This paper describes the process of development a Web application using Django framework. This application uses the Internet to facilitate communication among teachers and parents, and helps parents to track their children's grades and attendance in school.

Sadržaj

1. Uvod

Računari su sve više zastupljeni u savremenom društvu tako da se budući razvoj društva ne može zamisliti bez sve moćnijih računara. Potreba ljudi da brzo dođu do informacija iz raznih oblasti života uticala je na modernizaciju školstva i pojavu elektronskog dnevnika. On omogućava elektronsko evidentiranje ocena i izostanaka učenika i njihovo publikovanje putem Interneta. Elektronski dnevnik, takođe, poboljšava komunikaciju između nastavnika i roditelja, koji sada u svakom trenutku i sa bilo kog računara mogu videti obaveštenja iz škole. Roditelji su blagovremeno informisani o ponašanju i dostignućima učenika, čime im je omogućeno da stalno prate njihov rad i podstiču pravilan razvoj.

Ovaj rad opisuje korišćenje Django razvojnog okruženja za izradu školske administrativne aplikacije. Django je popularno razvojno okruženje, koje je napisano na programskom jeziku Python. Python je programski jezik visokog nivoa, prepoznatljiv po svojoj jednostavnoj i jasnoj sintaksi. Njegova dinamičnost i bogata standardna biblioteka skraćuju količinu programerskog koda i omogućavaju efikasan razvoj Web aplikacija uz pomoć Django razvojnog okruženja. Django koristi MVC (*Model View Controller*) obrazac za arhitekturu softvera, koji razdvaja aplikaciju na tri celine: model, pogled i kontroler. Time razdvaja logiku, prikaz podataka i pristup relacionoj bazi, čime omogućava lakšu saradnju programera jer svako obavlja svoj deo posla. Sadrži veliki broj šablona, koji se lako mogu promeniti, u zavisnosti od potreba aplikacije.

2. Uvod u Python

Python je programski jezik visokog nivoa, opšte namene, koji podržava proceduralni, funkcionalni i objektno-orijentisan stil programiranja, dajući programerima slobodu da odaberu pristup koji odgovara njihovom problemu. Razvoj jezika je počeo 1989. godine, dok je prva verzija objavljena 1991. godine.

Autor Python-a je Gvido van Rosum (Guido van Rossum), koji je u vreme nastanka jezika bio zaposlen na Nacionalnom istraživačkom institutu za matematiku i računarske nauke (*Centrum Wiskunde & Informatica*) u Amsterdamu. Proteklih sedam godina radio je u kompaniji Google, a u januaru ove godine prešao je u Dropbox. Kao iskusni programer kombinovao je najbolje karakteristike različitih programskih jezika želeći da napiše programski jezik koji će povećati produktivnost programera i omogućiti lakše čitanje koda. Inspiraciju za ime programskog jezika dobio je gledajući britansku humorističku seriju „Leteći cirkus Montija Pajtona" (engl. *Monty Python's Flying Circus*). Uprkos tome, zvanična maskota jezika postala je zmija piton ([1]).

Slika 1: Gvido van Rosum

2.1 Osobine Python-a

Python je zbog svoje jasne i jednostavne sintakse odličan za početnike u programiranju, ali je zbog svoje snage i prilagodljivosti privukao programere širom sveta. Neke od velikih kompanija koje koriste Python za svoje potrebe su Google, Yahoo, NASA, CERN, YouTube, IBM, Microsoft, Dropbox, Industrial Light & Magic, ITA… Google je koristio Python za svoj prvi *Web crawler* (Web robot ili Web pauk), program koji pretražuje Web stranice i analizira njihov sadržaj, a informacije se zatim šalju u bazu podataka. Instaler za Red Hat Enterprise Linux i Fedoru je napisan u Python-u i zbog toga je šaljivo nazvan Anakonda. Glavne odlike koje ga izdvajaju od drugih programskih jezika su:

- **Lako se uči i koristi.** Python je *jezik visokog nivoa*, što znači da je bliži govornom jeziku, nego mašinskom. Zbog vrlo čiste i jednostavne sintakse programi su čitljivi, a greške se lako uočavaju i ispravljaju. Time je povećana produktivnost profesionalnih programera jer su programi pisani na jeziku Python kraći i pišu se za manje vremena, nego programi pisani na mnogim drugim popularnim jezicima. Tip promenljive se ne deklariše, već se sam dodeli u toku interpretiranja. Promena tipa se eksplicitno izvršava funkcijama za konverziju ukoliko je to potrebno. Dobro je dokumentovan i postoji mnogo knjiga za učenje Python-a, kao i jaka zajednica korisnika koji se međusobno pomažu prilikom rešavanja problema. Za stalni razvoj jezika je zadužena neprofitna organizacija *Python Software Foundation (PSF),* koja se bavi normiranjem koda, ali i prikupljanjem donacija i organizovanjem konferencija ([2]).

- **Besplatan je i otvorenog koda.** Potpuno besplatno se instalira i koristi čak i u komercijalne svrhe. Može se kopirati, menjati i slobodno distribuirati, čime je omogućen stalni razvoj jezika.

- **Moćan je.** Python ima svu moć i prilagodljivost koja se očekuje od savremenog programskog jezika. Ima široku primenu i koristi se za najrazličitije namene. Razvijen je veliki broj standardnih modula koji omogućavaju efikasan rad u mnogim oblastima. Postoje moduli za kreiranje grafičkog korisničkog interfejsa, vezu ka relacionim bazama podataka, rad na leksičkoj analizi primenom regularnih izraza, kao i standardni moduli za mnoge druge poslove. Koriste ga i naučnici, na primer, za analizu gena, u matematičkim naukama za simulacije i optimizacije, za numeričku analizu, veštačku inteligenciju…

- **Prenosiv je.** Možemo ga koristiti bez obzira da li imamo Windows, Macintosh ili Linux, a Python podržavaju i mnoge druge platforme kao što su FreeBSD, Java, Solaris, Symbian OS, AIX operating system, Amiga, AROS, AS/400, BeOS, OS/2, OS/390, Palm OS, Plan 9, PlayStation 2, Psion, QNX, RISC OS (ranije Acorn), Sharp Zaurus, VMS, VxWorks, Windows CE/Pocket PC, Xbox, z/OS ([3]) ... Program napisan na Python-u će raditi na bilo kom računaru na kojem je instaliran Python, bez obzira na operativni sistem.

- **Python je interpretirani jezik.** Python programi se izvršavaju direktno od strane interpretatora, bez prevođenja na mašinski jezik. Datoteka koja se interpretira izvršava se redno liniju po liniju. Izvorni kôd se prevodi u tzv. bajt-kod, koji je arhitektonski neutralan i koji zatim interpretator izvršava na korisničkom računaru. Bajt-kod se može čuvati u `*.pyc` datotekama i kasnije koristiti, čime se preskače stalno prevođenje. Ovo doprinosi prenosivosti programa i brzini programiranja. Uz Python se isporučuje i integrisano razvojno okruženje pod nazivom IDLE. U interaktivnom režimu rada prevodi se linija po linija koda sa trenutnim odzivom. Rezultat je odmah vidljiv, što je odlično za učenje jezika, ali i za testiranje novih ideja i delova koda. Postoji i skript režim rada koji je pogodan za pisanje većih programa, koji mogu da se čuvaju i menjaju.

- **Objektno-orijentisan je.** Prilikom projektovanja računarskih programa najčešće se kao osnova koriste objekti, a Python podržava i koncepte objektno-orijentisanog programiranja, kao što su nasleđivanje i polimorfizam. To je za obimnije programe najbolji način pisanja koda jer omogućava ponovno korišćenje koda. Kod pisanja jednostavnijih programa nije neophodno koristiti objekte, zato što Python podržava i proceduralni i funkcionalni stil programiranja. Time se „Zdravo, svete“ svodi na liniju:

  ```
  print("Zdravo, svete")
  ```

  Upravo ova fleksibilnost odabira paradigme čini Python tako interesantnim programskim jezikom. Može biti vrlo jednostavan i poslužiti početnicima, ali i moćan jezik koji podržava objektno-orijentisani stil programiranja.

- **Automatski upravlja memorijom.** Python ima ugrađeni *garbage collector* koji traži objekte koji se više ne koriste u programu. Ukoliko ih nađe, on automatski uklanja te objekte i oslobađa memoriju. Ovim je pisanje softvera značajno

olakšano, programeri ne moraju nepotrebno da se bave upravljanjem memorijom, već imaju više vremena za rešavanje konkretnog problema.

- **Odlično se integriše sa drugim jezicima i alatima.** Python je poznat kao *glue language,* odnosno ponaša se kao lepak kojim se mogu povezati delovi koda koji su napisani na drugim programskim jezicima, kao što je C, C++ ili Java ([1]). Ukoliko je potrebno da se neki delovi programa brže izvršavaju, mogu se koristiti ovi jezici. Python se koristi i kao prototipski jezik. Zahvaljujući njegovoj jednostavnoj sintaksi, programi se najpre implementiraju i testiraju u Python-u, a zatim u celosti prevedu u konačan jezik.

Osnovna i najčešće korišćena implementacija Python-a napisana je na jeziku C, a često se u literaturi naziva i CPython. Kako bi se prednosti ovog jezika što bolje upotrebile, vremenom su se pojavile nove verzije za druge programske jezike. Kompanija Microsotf pomaže razvoj implementacije u C#-u na .NET platformi pod nazivom IronPython, dok Sun potpomaže razvoj implementacije Python-a u Javi koja se zove Jython i izvršava na Java virtualnoj mašini (JVM). Jython programi mogu da uvoze i koriste bilo koje Java klase, umesto standardnih Python modula. PyS60 je Python implementacija za Symbian platformu i koristi ga kompanija Nokia za programiranje nekih modela mobilnih telefona. Ovo su samo neke od implementacija koje se često primenjuju.

2.2 Razvoj Python-a

Postoje dve grane ovog jezika, a to su Python 2.x i Python 3.x. Verzija Python 2.0 je objavljena 2000. godine, a veliki iskorak se desio 2008. kada je izašla verzija Python 3.0, poznata i kao Python 3000 i Py3K. Gvido van Rosum je tada odlučio da unese dosta promena, želeći da otkloni neke nedostatke i doprinese čistoći jezika. Svaka Python 2.x verzija je kompatibilna sa prethodnim verzijama i omogućava korišćenje velikog broja napisanih modula, dok je Python 3.0 prekinuo kompatibilnost sa već postojećim verzijama. Neke od novosti koje su uvedene su: **`print()`** nije više naradba, već funkcija, pa je potrebno pisati argumente u zagradama, kao na primer:

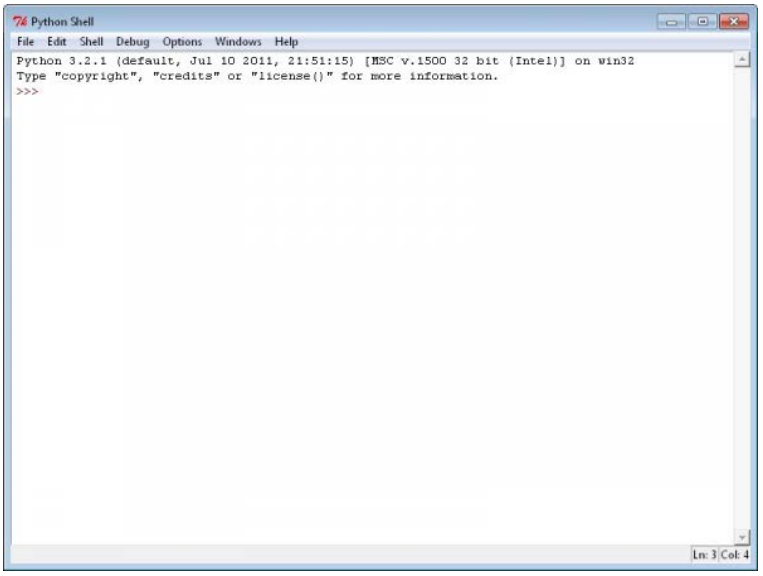
`print "Zdravo, svete"` je postalo **`print ("Zdravo, svete")`**,

zatim celobrojni tip **`long`** je preimenovan u **`int`**, odnosno programeri ne moraju voditi računa o veličini brojeva; funkcija **`raw_input()`**, koja prima tekst koji korisnik upiše, u verziji Python 3.x postaje **`input()`**; funkcije **`map()`** i **`filter()`** vraćaju iteratore umesto listi, kao i **`range()`** koja nastaje od funkcije **`xrange()`**; sve niske su Unicode…

Aktivno se radi na razvoju Python 3.x verzija i pisanju novijih biblioteka koje su za njih dostupne. Uprkos tome, mnogi korisnici i dalje koriste neke od verzija Python 2.x, zbog velike količine koda koji mogu da upotrebe. Trenutno su dostupne verzije Python 2.7.5 i Python 3.3.2 koje su objavljene u maju ove godine.

2.3 Instalacija Python-a i razvojno okruženje

Python interpretator je sastavni deo većine Linux/Unix platformi, a korisnici Windows i drugih operativnih sistema moraju ga dodatno instalirati. Instalacija se može besplatno preuzeti sa zvanične Internet stranice Python-a, gde su dostupne najnovije verzije jezika ([4]). Detaljan opis postupka instaliranja se može videti u [1]. Instalacija uključuje i IDLE (skraćenica od **I**ntegrated **D**eve**L**opment **E**nvironment, ali je Idle i ime jednog od članova Monti Pajton trupe), prvo integrisano razvojno okruženje za Python, koje se sastoji od skupa alata koji olakšavaju programiranje. Kada se pokrene IDLE prikazuju se sledeći prozor:



Slika 2: Python Shell, IDLE u interaktivnom režimu rada

Kada je ovaj prozor aktivan, korisnik se nalazi u Python Shell režimu rada, to jest u interaktivnom režimu rada. Upisuju se naredbe i neposredno potom vide rezultati. Trostruka strelica (**>>>**) je znak da je Python spreman za unos naredbe. Tradicija je da se kao prvi program u knjigama o programiranju navodi ispisivanje poruke „Hello, world!“. U Python-u je to veoma lako, dovoljno je napisati **`print("Hello, world!")`** iza trostrukih strelica i pritisnuti **Enter**. U drugom redu će se ispisati očekivani rezultat.

```
>>> print ("Hello, world!")
 Hello, world!
```

Različitom bojom su ispisani različiti delovi koda, čime ovo okruženje pomaže da se lakše sagleda kôd i uoče greške, što je naročito bitno kod kompleksnijih programa.

Za pisanje programa, koji će se čuvati i kasnije izvršavati, koristi se skript režim rada. Programi se pišu u posebnom prozoru koji se može otvoriti iz interaktivnog prozora kada se iz menija **File** izabere opcija **New Window**. Pre izvršavanja programa neophodno ih je snimiti sa ekstenzijom **`.py`**. Program se pokreće klikom na meni **Run**, a zatim opciju **Run Module.** Prečica za ovu akciju je taster **F5** na tastaturi. Rezultat rada programa će se pokazati u interaktivnom prozoru. Već napisani program se može ponovo pokrenuti tako

što se dva puta klikne mišem na ikonicu programa. Moguće je menjati program tako što se desnim tasterom miša klikne na ikonicu programa i izabere opcija **Edit with IDLE**.

Ukoliko naiđe na grešku, interpretator će ispisati poruku. Treba obratiti pažnju na to da je Python jezik osetljiv na veličinu slova *(case-sensitive)*, tako da se u njemu pravi razlika između velikih i malih slova. Ukoliko se napiše **`Print`** interpretator to neće prepoznati kao funkciju za štampanje.

```
>>> Print("Hello, world!")
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    Print("Hello, world!")
NameError: name 'Print' is not defined
```

Komentari se u Python-u pišu iza simbola **#**. Sve ono što se nalazi u istoj liniji iza ovog simbola, interpretator zanemaruje. Namenjeni su programerima i mogu biti velika olakšica prilikom čitanja i analiziranja koda. U IDLE okruženju komentari su istaknuti crvenom bojom da bi bili uočljiviji.

3. Opis jezika

3.1 Promenljive i tipovi podataka

U programskom jeziku Python podaci su predstavljeni objektima. Svaki podatak je predstavljen objektom ili relacijom između objekata. Objekati mogu imati četiri karakteristike:

- identitet (referenca) – adresa u memoriji na kojoj se nalazi podatak
- tip – veličina prostora u memoriji gde se nalazi podatak
- vrednost (atribut) – sadržaj koji se nalazi na toj memorijskoj lokaciji
- metode – funkcije pridružene objektu, koje opisuju ponašanje.

Objekti u Python-u mogu biti ugrađeni (engl. *built-in objects*), a mogu se generisati i pomoću klasa koje programeri pišu za svoje potrebe. **Promenljive** (engl. *variables*) služe za skladištenje i obradu podataka. Kada se kreira neka promenljiva, tada se rezerviše neki prostor u memoriji. Kasnije se podacima pristupa preko imena promenljive, bez potrebe da se zna gde se tačno u memoriji nalazi traženi podatak. Prilikom definisanja imena promenljivih potrebno je voditi računa o tome da sadrže samo slova, brojeve i znakove za podvlačenje (_), kao i da ime promenljive ne može početi brojem. U tradiciji jezika Python je da imena promenljivih počinju malim slovom i izbegava se upotreba znaka za podvlačenje na početku. Za dodeljivanje vrednosti promenljivima koristi se operator (**=**). Naredba za dodeljivanje vrednosti najčešće dodeljuje vrednost nekoj promenljivoj, a ako promenljiva još ne postoji, naredba je najpre generiše, a zatim joj dodeli vrednost.

```
>>> ime = 'Python'
```

Prethodnom naredbom definiše se promenljiva sa imenom **`ime`** i dodeljuje joj se vrednost koja je referenca na znakovni niz **`'Python'`**. Ne mora se deklarisati tip promenljive, već Python sam dodeljuje tip u toku implementiranja na osnovu vrednosti promenljive. Sve vrednosti promenljivih su objekti.

Jednom kreirani objekat ne može da menja tip, dok vrednost objekata može da bude promenjena. Promenljivost objekata određena je njihovim tipom. Postoje promenljivi i nepromenljivi objekti. U nepromenljive objekte spadaju brojevi, niske (engl. *strings*) i n-torke, dok su rečnici i liste promenljivi objekti ([3]). Detaljan opis ovih tipova podataka i funkcija za rad sa njima dat je u knjizi [5].

3.2 Kontrola toka izvršavanja

Python sadrži naredbe za kontrolu toka koje su uobičajene za većinu programskih jezika. Specifičan je po tome što blokove koda odvaja uvlačenjem koda u odnosu na rezervisanu reč naredbe udesno. Uvlačenjem se redovi grupišu ne samo vizuelno, već i logički, jer zajedno čine jedan logički blok. Ovo označavanje bloka iskaza uvlačenjem koristi se kod svih kontrolnih struktura. Tako, na primer, oblik **`if`** naredbe glasi:

```
if uslov:
      kôd koji se izvršava ukoliko je uslov zadovoljen
else:
      kôd koji se izvršava ukoliko je uslov nije zadovoljen
```

Opis svih naredbi za kontrolu toka izvršavanja, kao i primeri njihovih korišćenja, dati su u knjizi [6].

3.3 Funkcije

Do sada je bilo reči o funkcijama koje su ugrađenje u Python, na primer: **`print()`**, **`range()`** ili **`input()`**. Funkcija **`range(broj)`** vraća sekvencu brojeva, počev od 0 do prvog broja koji je manji od zadatog broja. Često se koristi u iteracijama. Funkcija **`input()`**, odnosno **`raw_input()`** u verzijama Python 2.x, prima tekst koji korisnik upiše. Argument funkcije **`input()`** je znakovni niz koji poziva korisnika da upiše svoj tekst. Kada korisnik unese tekst i pritisne taster **Enter**, funkcija vraća ono što je korisnik otkucao u obliku znakovnog niza i dodeljuje odgovarajućoj promenljivoj. Neke od funkcija koje se takođe često koriste su i funkcije za konverziju tipova. Na primer, funkcija **`int(x)`**, koja vraća celobrojnu vrednost koja se dobija konverzijom niske ***x*** u ceo broj, kao i inverzna operacija **`str(x)`**, koja vraća znakovnu vrednost koja se dobija konverzijom celog broja ***x*** u nisku.

3.3.1 Pisanje korisnički definisanih funkcija

Potpuno isto rade i funkcije koje korisnici sami napišu. Mogu se pozivati više puta i omogućavaju da se kôd raspodeli u manje celine kojima se lakše upravlja. Definicija funkcije počinje rezevisanom reči `def` iza koje slede ime funkcije i par malih zagrada sa ulaznim parametrima. Funkcija može biti bez parametara, a može imati i jedan ili više parametara, koji su odvojeni zarezima. Svi parametri se prenose po referenci i promene njihovih vrednosti su vidljive i izvan funkcije. Blok naredbi počinje dvotačkom. Niz naredbi koji slede naziva se telo funkcije. Funkcije imaju i poseban mehanizam koji omogućava da se dokumentuju pomoću dokumentacijskog znakovnog niza (engl. *documentation string*, ili kraće *docstring*). Najčešće je ograničen trostukim navodnicima i piše se u prvom redu funkcije. Nije neophodan, ali je korisno napisati kraći opis kako funkcija radi. Funkcija vraća vrednost pomoću iskaza **`return`**. Može imati jedan ili više iskaza **`return`**, a može postojati i bez njega. Primeri funkcija sa različim brojem iskaza **`return`** dati su u knjizi [7]. Funkcija se poziva tako što se napiše ime funkcije i u zagradama navede lista argumenata.

```
def funkcija (parametar1, parametar2, …):
      """ Dokumentacijski znakovni niz.  """
      telo funkcije
      return [vrednost, ukoliko funkcija vraća vrednost]
```

Sledi primer jednostavne funkcije, koja služi za računanje srednje vrednosti elemenata liste.

```
>>> def srednjaVrednost (lista):
      " " " Funkcija nalazi srednju vrednost elemenata liste." " "
      zbir = 0
      for x in lista:
            zbir +=x
      return zbir/len(lista)
>>> srednjaVrednost ([2,5,7,4,12])
6.0
```

3.3.2 Parametri funkcija

Parametri su imena promenljivih koja se navode unutar zagrada u zaglavlju funkcije. Preuzimaju vrednosti koje se pri pozivanju funkcije prosleđuju preko njenih parametara. Ako se u zaglavlju funkcije navede samo lista parametara, na taj način su definisani pozicioni parametri. Ukoliko se funkcija pozove samo sa nizom vrednosti, zadati su pozicioni argumenti. Upotreba pozicionih parametara i pozicionih argumenata znači da parametari dobijaju vrednosti samo na osnovu položaja vrednosti koje im se prosleđuju. Prvi parametar dobija prvu prosleđenu vrednost, drugi drugu prosleđenu vrednost, i tako dalje. U sledećem primeru parametar **ime** dobija vrednost **'Marko'**, a parametar **godine** vrednost **16**.

```
>>> def rođendan (ime, godine):
      print ('Srećan rođendan,', ime,'! Sada imaš', godine,
'godina!')
>>> rođendan ('Marko', 16)
Srećan rođendan, Marko ! Sada imaš 16 godina!
```

Pozicioni parametri dobijaju vrednosti redosledom koji je zadat, osim kada se u pozivu funkcije zada drugačije. Funkciji je moguće eksplicitno zadati vrednosti parametara, bez obzira na redosled, ako se argumenti funkcije upotrebe kao ključne reči. Tako sledeći poziv prethodne funkcije rezultuje istom porukom.

```
>>> rođendan (godine = 16, ime = 'Marko')
Srećan rođendan, Marko ! Sada imaš 16 godina!
```

Argumenti upotrebljeni kao ključne reči omogućavaju prosleđivanje vrednosti proizvoljnim redosledom, ali i doprinose jasnoći koda. Takođe, u Python-u postoji mogućnost da se parametrima dodele podrazumevane vrednosti, to jest vrednosti koje parametri preuzimaju kada im nije zadata nikakva vrednost. Ukoliko bi se izmenilo zaglavlje prethodne funkcije na sledeći način:

```
>>> def rođendan2 (ime = 'Marko' , godine = 16):
      print ('Srećan rođendan,', ime,'! Sada imaš', godine,
'godina!')
```

to bi značilo da ako parametru **ime** nije zadata vrednost, on preuzima vrednost **'Marko'**. Parametar **godine** će preuzeti vrednost **16** ukoliko nije zadato drugačije. Zato sledeći poziv **rođendan2()** neće izazvati grešku, nego vraća već poznatu poruku.

Sledećim pozivom funkcije:

```
>>> rođendan2 (ime = 'Sofija')
Srećan rođendan, Sofija ! Sada imaš 16 godina!
```

parametar **ime** je dobio vrednost **'Sofija'**, a parametar **godine** ima svoju podrazumevanu vrednost. Podrazumevane vrednosti su odlično rešenje za funkcije koje imaju istu vrednost jednog ili više parametara u svakom pozivu.

Broj ulaznih parametara može biti promenljiv, što se obeležava znakom * ispred naziva ulaznog parametra. Sledeća funkcija izračunava zbir svih brojeva koji su zadati kao vrednosti parametara.

```
>>> def zbir (*brojevi):
      rezultat = 0
      for i in brojevi :
            rezultat += i
      return rezultat
>>> zbir (27, 32, 15)
74
```

3.3.3 Opseg važenja

Opsezi važenja predstavljaju oblasti programa koje su odvojene jedna od druge. Globalni opseg važi izvan svih funkcija, dok je lokalni opseg prostor za identifikatore unutar definicije funkcije. Sve promenljive koje se definišu unutar neke funkcije su lokalne za tu funkciju i ne mogu im pristupati naredbe iz drugih delova programa. Ukoliko dve promenljive imaju isto ime unutar dve zasebne funkcije, to su dve potpuno različite promenljive i nemaju nikakvog uticaja jedna na drugu. Ako se promenljivoj unutar funkcije dodeli ime globalne promenljive, tada lokalna promenljiva prekriva globalnu. Promene vrednosti lokalne promenljive neće uticati na vrednost globalne promenljive, i o tome se mora voditi računa da ne bi bilo zabune. Zbog toga se ne preporučuje korišćenje imena globalnih promenljivih prilikom definisanja promenljivih unutar funkcija. Da bi se dobio potpuni pristup globalnoj promenljivoj, pa samim tim i mogućnost promene vrednosti, potrebno je navesti ključnu reč global ispred naziva promenljive unutar funkcije.

3.4 Klase i objekti

Python podržava objektno-orijentisan stil programiranja i omogućava da se objekti iz stvarnog sveta predstave u obliku softverskih objekata. Objekti se instanciraju na osnovu klasa, koje predstavljaju nacrt za izradu objekata. Klase sadrže promenljive, a gotovo uvek sadrže i metode, odnosno operacije koje objekat može da uradi.

3.4.1 Definicija klase

Klasa se definiše upotrebom ključne reči **`class`**. Zatim sledi ime klase, koje po uobičajenoj konvenciji počinje velikim slovom. Ukoliko je klasa nasleđena iza imena se piše naziv natklase u malim zagradama. Osnovni ugrađeni Python-ov tip je **`object`**, i nova klasa se može zasnivati ili na klasi **`object`** ili na nekoj drugoj prethodno definisanoj klasi. Sledeći red može biti dokumentacijski znakovni niz koji dokumentuje klasu i opisuje vrstu objekata koji se mogu praviti od klase.

```
class ImeKlase[(KlasaRoditelj)]:
      " " " Opcioni dokumentacijski znakovni niz." " "
      klasna_promenljiva = vrednost
      def metoda(self, ...)
            telo metode klase
```

Metode su funkcije koje su pridružene određenom objektu. U Python-u svaka metoda ima specijalni prvi parametar, koji se po konvenciji zove **`self`**. Taj parametar omogućava metodi da referencira sam objekat. To znači da preko parametra **`self`** metoda može doći do objekta koji ju je pozvao i pristupiti njegovim atributima i metodama. Pojedinačni primerci klase se kreiraju upotrebom funkcijske notacije, na primer **`x = ImeKlase()`**, kada se potpuno novi objekat ove klase dodeljuje promenljivoj **`x`**. Metoda ovog objekta se može pozvati pomoću notacije sa tačkom, **`x.metoda()`**.

3.4.2 Konstruktori

U prvom delu koda klase obično se definiše i specijalna metoda, nazvana konstruktor, koja se automatski poziva čim se generiše novi objekat. Konstruktorska metoda se najčešće koristi za zadavanje početnih vrednosti atributa objekta. Ova metoda se, po pravilu, naziva **`__init__`**. Pošto je ovo jedna od specijalnih metoda u Python-u, ime počinje i završava se sa dva znaka za podvlačenje.

```
>>> class Krug:
      pi = 3.141592
      def __init__(self, poluprečnik):
            self.poluprečnik = poluprečnik
      def površina(self):
            return print ("Površina kruga je:", self.poluprečnik *
self.poluprečnik * Krug.pi)
>>> k = Krug(5)
>>> k.površina()
Površina kruga je: 78.5398
```

3.4.3 Nasleđivanje

Python u potpunosti podržava jedan od najkorisnijih koncepata objekto-orijentisanog programiranja – nasleđivanje. Nova klasa automatski nasleđuje sve metode i

atribute postojeće klase. Nasleđivanje je naročito korisno kada treba da se napravi uže specijalizovana verzija postojeće klase. Tada se novoj klasi dodaju nove metode i atributi da bi se proširila funkcionalnost objekata nove klase. Originalna klasa se zove osnovna klasa ili natklasa (engl. *base class*), a nova je izvedena klasa ili potklasa (engl. *derived class*). Svaka klasa može da nasledi jednu ili više drugih klasa. Definicija takve klase može izgledati ovako:

```
class ImeKlase (OsnovnaKlasa1, OsnovnaKlasa2, OsnovnaKlasa3,…):
      iskaz 1
      iskaz 2
      …
      iskaz n
```

U zagradi navedena **OsnovnaKlasa1**, **OsnovnaKlasa2**, **OsnovnaKlasa3**,… su imena klasa iz kojih je izvedena nova klasa sa imenom **ImeKlase**. Najčešće se nasleđivanje javlja u formi jednostrukog nasleđivanja sa jednom osnovnom klasom. Ukoliko se radi o višestrukom nasleđivanju, pri pristupanju jednom atributu klase njegovo ime će se prvo tražiti u izvedenoj klasi **ImeKlase**, a ako se ne nađe tu, tražiće se u **OsnovnaKlasa1** i u svim klasama koje su njena natklasa. Tek ako ne bude tu pronađeno preći će se na **OsnovnaKlasa2**. Ako se u izvedenoj klasi definišu promenljive ili metode koje imaju isti naziv kao u osnovnoj klasi, objekti izvedene klase će koristiti promenljive ili metode izvedene klase. Ukoliko je potrebno koristiti neke atribute natklase, navodi se celo ime, na primer, **OsnovnaKlasa1.atribut**. Takođe je moguće redefinisati kako će nasleđena metoda osnovne klase raditi u izvedenoj klasi. To je poznato kao redefinisanje (engl. *overriding*) metode.

3.4.4 Enkapsulacija

Koncept enkapsulacije u Python-u omogućava da se određenim objektima ne može pristupati izvan klase u kojoj su definisani. Svi atributi i metode objekata standardno su javni, ali se mogu definisati i kao privatni, što znači da im samo druge metode samog objekta mogu lako pristupati ili ih pozivati. Kako bi se sprečilo da klijenti direktno pristupaju atributima objekata, koriste se **privatni atributi**. Definišu se tako što ime atributa započinje sa dva znaka za podvlačenje (__). Pristupa im se unutar definicije klase objekata, a pokušaj da mu se pristupi izvan definicije klase bi izazvao grešku. **Privatne metode** se definišu na isti način, dodavanjem dva početna znaka za podvlačenje u ime metode. Lako im pristupaju sve druge metode unutar iste klase.

```
>>> class MojaKlasa:
        def __privatna_metoda(self):
                        print ("Ovo je privatna metoda.")
                def javna_metoda(self):
                        print ("Ovo je javna metoda.")
                self.__privatna_metoda()
>>> c = MojaKlasa ()
>>> c.javna_metoda ()
```

Javna metoda prikazuje na ekranu znakovni niz **"Ovo je javna metoda."**, a zatim poziva privatnu metodu objekta. Privatna metoda takođe ispisuje znakovni niz, pa je rezultat rada ovog programa:

```
Ovo je javna metoda.
Ovo je privatna metoda.
```

Pokušaj da se na isti način pristupi privatnoj metodi objekta **c**, pišući **c.privatna_metoda()** ili **c.__privatna_metoda()**, izazvaće poruku o grešci.

3.5 Moduli

Moduli su datoteke koje sadrže kôd koji je najčešće namenjen za korišćenje u drugim programima. U njima je grupisan kôd koji se odnosi na određenu funkcionalnost. Kreiranje dužih programa sastoji se obično iz pisanja modula, koji mogu da budu grupisani u pakete (engl. *packages*). Postoji veliki broj biblioteka modula koji su deo Python-ove instalacije, ali velike prednosti ima i izrada vlastitih modula. Omogućavaju višestruku upotrebu koda, što doprinosi efikasnosti pisanja programa. Takođe, kada se programi podele na logičke module, lakše se upravlja velikim programima. Razdvajanje koda u module olakšava timski rad, jer svaki član radnog tima može da se bavi različitim delom projekta. Vlastiti moduli se mogu slati i deliti sa drugim programerima, koji ga mogu koristiti na sličan način kao i svaki ugrađen Python-ov modul.

3.5.1 Pisanje modula

Moduli se prave na isti način kao i svaki drugi program u Python-u. Trebalo bi da se definiše grupa programskih komponenata, kao što su funkcije i klase i sačuva se u datoteci sa ekstenzijom **.py**. U sledećem primeru napisan je jednostavan modul sa dve funkcije. Prva funkcija uvećava, a druga umanjuje zadati broj za jedan, ukoliko parametru **korak** nije dodeljenja vrednost. U suprotnom, broj će biti uvećan, odnosno umanjen, za zadatu vrednost parametra **korak**. Ime modula je **primer_modula**, jer je kôd sačuvan u datoteci pod nazivom **primer_modula.py**.

```
def uvećaj (x, korak = 1):
    return x + korak

def umanji (x, korak = 1):
    return x - korak
```

3.5.2 Uvoženje modula

Modul se uvozi pomoću naredbe **import**. Da bi se uvezao ceo modul, dovoljno je iza reči **import** napisati naziv modula. Ukoliko se učitava samo deo modula, koristi se reč **from**.

```
from ime_modula import ime_dela_modula
```

Kreirani modul **primer_modula** se sada može učitati u interaktivnom režimu rada i mogu se koristiti njegove funkcije na sledeći način:

```
>>> import primer_modula
>>> primer_modula.umanji (5)
4
>>> primer_modula.uvećaj (23, 5)
28
```

3.5.3 Standardni moduli

Standardni moduli u velikoj meri proširuju mogućnosti jezika. Na primer, ako je potrebno upotrebiti neku matematičku funkciju ili konstantu, uvozi se matematičku modul **math**. Funkcija **dir()** pruža uvid u spisak svih imena koji su dostupni u nekom modulu.

```
>>> import math
>>> dir (math)
['__doc__', '__name__', '__package__', 'acos', 'acosh', 'asin',
'asinh', 'atan', 'atan2', 'atanh', 'ceil', 'copysign', 'cos',
'cosh', 'degrees', 'e', 'exp', 'fabs', 'factorial', 'floor',
'fmod', 'frexp', 'fsum', 'hypot', 'isinf', 'isnan', 'ldexp',
'log', 'log10', 'log1p', 'modf', 'pi', 'pow', 'radians', 'sin',
'sinh', 'sqrt', 'tan', 'tanh', 'trunc']
>>> math.pi
3.141592653589793
>>> math.sin (math.pi/2)
1.0
```

Koristeći reč **from** moguće je uvesti samo funkcije i konstante koje su potrebne za rešenje određenog zadatka. U tom slučaju se ne koristi notacija sa tačkom za pristup i ne navodi naziv modula.

```
>>> from math import sin, pi
>>> sin (pi/2)
1.0
```

Veliki broj Python programera svakodnevno radi na razvoju mnogih standardnih modula koji olakšavaju pisanje koda i omogućavaju efikasniji rad. Ovo je samo jedna od prednosti zbog koje se programeri odlučuju da svoje aplikacije razvijaju koristeći ovaj jezik.

4. Django razvojno okruženje

Svakodnevna potreba za pisanjem Web aplikacija dovela je do razvoja velikog broja razvojnih okruženja koja omogućavaju lakši i brži razvoj ovih aplikacija. Django je jedno od najpopularnijih razvojnih okruženja za razvoj Web aplikacija koje je pisano na programskom jeziku Python. Sadrži gotove alate za realizaciju postupaka koji se često ponavljaju u aplikacijama. Time se problemi vezani za autentikaciju i autorizaciju, povezivanje sa bazom ili pisanje administratorske stranice, rešavaju vrlo efikasno uz malo truda i uloženog vremena.

4.1 Razvoj Django okruženja

Django je počeo da se razvija 2003. godine kada su programeri Adrijan Holovati i Sajmon Vilson odlučili da, umesto PHP-a, koriste Python za izradu aplikacija. Bili su zaposleni u Lawrence Journal-World novinama u Kanzasu i svakodnevno su se bavili uređenjem sajtova za objavljivanje najnovijih vesti. Kako je njihov posao zahtevao brz i jednostavan razvoj aplikacija, tragali su za okruženjem koje će im to omogućiti. Njihovom timu se priključuje i Džejkob Kaplan-Mos i, nakon dve godine rada, u julu 2005. godine objavljuju razvojno okruženje Django. Inspiraciju za naziv su dobili slušajući džez gitaristu i kompozitora Đanga Rajnharta ([8]). Ovo okruženje je brzo steklo popularnost i sada, nakon nekoliko godina, ima na hiljade korisnika širom sveta. Neki od poznatih sajtova koji koriste Django su Pinterest, Instagram, Mozilla i The Washington Times ([9]).

4.2 Potreban alat za kreiranje aplikacije

Na zvaničnoj Internet stranici Django razvojnog okruženja nalazi se materijal i detaljno uputstvo za njegovu instalaciju ([10]). Poslednja zvanična verzija je Django 1.5.2 koja je objavljena u avgustu ove godine. Podržane baze podataka su MySQL, PostgreSQL, Oracle i SQLite, a potrebno je instalirati i Python modul za rad sa bazom podataka.

Kreiranje projekta može započeti iz komandne linije, na sledeći način: pozicioniranjem u direktorijum gde projekat treba da bude sačuvan i ukucavanjem komande **`django-admin.py startproject moj_sajt`**. Ovim je kreiran direktorijum pod nazivom **`moj_sajt`** sa sledećom strukturom:

```
moj_sajt/
        manage.py
        moj_sajt/
                  __init__.py
                 settings.py
                  urls.py
                  wsgi.py
```

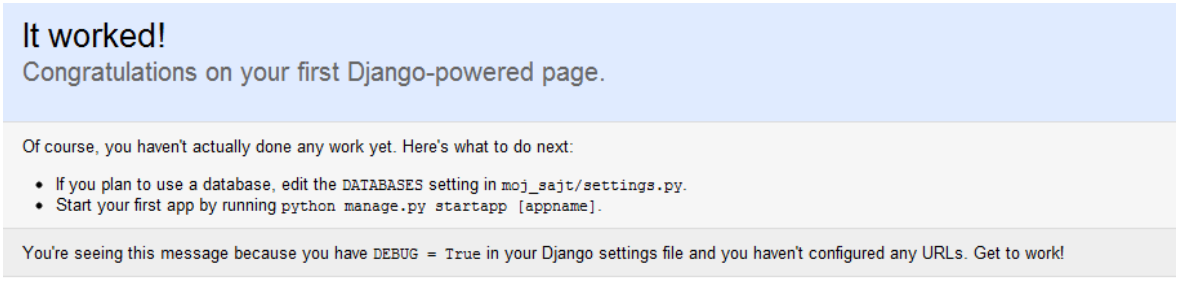
Datoteka **manage.py** omogućava komunkaciju sa projektom, to jest pokretanje akcija kao što su: **runserver** za pokretanje servera, ili **syncdb** za osvežavanje baze podataka. Datoteka **moj_sajt/__init__.py** je prazna i označava da je direktorijum Python paket. U datoteci **moj_sajt/settings.py** se nalaze podešavanja projekta, a u datoteci **moj_sajt/urls.py** spajanje url-ova i pogleda koji se kreiraju u aplikaciji. Datoteka **moj_sajt/wsgi.py** sadrži podešavanja za Web server.

Da bi se pokrenuo kreirani projekat, potrebno je pozicionirati se preko komandne linije u direktorijum **moj_sajt** i uneti komandu **python manage.py runserver**. Ukoliko je pokretanje uspešno izvršeno ispisaće se:

```
Validating models...

0 errors found
Django version 1.4, using settings 'moj_sajt.settings'
Development server is running at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[19/Aug/2012 09:48:53] "GET / HTTP/1.1" 200 1959
```

Na adresi **http://127.0.0.1:8000/** u Internet pretraživaču će se pojaviti poruka da je server uspešno pokrenut.



Slika 3: Poruka koja se ispisuje u Internet pretraživaču

Za kreiranje aplikacije u projektu potrebno je nalaziti se u direktorijumu projekta i pokrenuti komandu **python manage.py startapp moja_aplikacija**. Tada se kreira direktorijum **moja_aplikacija** sa sledećim datotekama:

```
moja_aplikacija /
        __init__.py
       models.py
       tests.py
       views.py
```

4.3 MVC obrazac

Django koristi MVC (*Model View Controller*) obrazac za arhitekturu softvera, koji razdvaja aplikaciju na tri celine: model, pogled i kontroler. Model predstavlja podatke koji se koriste u aplikaciji, pogled prikazuje podatke korisnicima, a kontroler omogućava komunikaciju modela i pogleda i sadrži logiku aplikacije. Django razdvaja

logiku, prikaz i pristup bazi, ali koristi drugačije nazive za ove celine. Celinu za prikaz naziva *template*, dok deo koji je kontroler naziva *view*, tako da se ovaj obrazac za arhitekturu ovde naziva i MTV (*Model Template View*)([11]). U datoteci **models.py** pišu se klase koje nasleđuju klasu **Model** i koje predstavljaju tabele u bazi, a atributi klasa su kolone u tabeli. Sledi primer jednostavnog modela koji oslikava tabelu **Knjiga** sa kolonama **naziv** i **ime_autora**, gde je maksimalna dužina unosa 50 karaktera, **datum_objavljivanja** tipa **DateTimeField**, dok atribut **izdava**c može biti null i nije obavezan unos.

```
from django.db import models

class Knjiga (models.Model):
    naziv = models.CharField(max_length=50)
    ime_autora = models.CharField(max_length=50)
    datum_objavljivanja = models.DateTimeField()
    izdavac = models.CharField(max_length=50, null=True,
blank=True)
```

Ime baze i parametre za povezivanje sa bazom je potrebno uneti u datoteku **settings.py**.

```
DATABASES = {
      'default': {
      'ENGINE':   'django.db.backends.postgresql_psycopg2',   #Tip
#baze je PostgreSQL
      'NAME': 'Knjige',   #Ime baze.
      'USER':  'postgres',  #Korisničko ime i lozinka za pristup
#bazi.
      'PASSWORD': 'admin',
      'HOST': ' ',   #Prazna niska za localhost.
      'PORT': ' ',   #Broj porta na kom je instalirano.
      }
}
```

Kako bi Django iskoristio napisani modele pri kreiranju baze, potrebno je dodati ime aplikacije u n-torku **INSTALLED_APPS**, koja se nalazi u datoteci **settings.py** i sadrži imena svih aplikacija koje se mogu koristiti u tom projektu.

```
INSTALLED_APPS = (
      #...
      'moja_aplikacija',
      #...
)
```

Komandom **python manage.py syncdb** kreirana je tabela **Knjiga** sa navedenim kolonama. Prilikom prve sinhronizacije sa bazom, kreira se korisnik sa administratorkim privilegijama (*superuser*) i dodeljuje mu se lozinka za pristup aplikaciji.

Logika aplikacije je smeštena u datoteci **views.py**, gde se definišu funkcije i pozivaju HTML stranice za prikaz podataka.

```
from django.shortcuts import render_to_response
from models import Knjiga

def najnovije_knjige(request):
    lista_knjiga = Knjiga.objects.order_by(
'- datum_objavljivanja')[:10]
    return render_to_response('najnovije_knjige.html',
{'lista_knjiga': lista_knjiga})
```

Funkcija **najnovije_knjige()** sortira knjige po datumu objavljivanja, počev od najnovije, izdvaja prvih deset i poziva stranicu **najnovije_knjige.html**. Datoteka **views.py** može sadržati više funkcija, a koja od njih će biti pozvana definiše se u datoteci **urls.py**. Ovde se vrši odabir URL-ova aplikacije, na primer:

```
from django.conf.urls.defaults import *
import views

urlpatterns = patterns(' ',
    ( r'^najnovije/$', views.najnovije_knjige ),)
```

Prvi parametar je regularni izraz, gde **r** na početku označava da nije moguće preskočiti nijedan znak pri čitanju URL-ova. Ukoliko se u Internet pretraživaču ukuca http://127.0.0.1:8000/najnovije/, poziva se funkcija **najnovije_knjige()**. Na stranici **najnovije_knjige.html** se definiše obrazac za prikazivanje podataka. Ovi obrasci sadrže HTML i Django tagove. Između tagova **"{{"** i **"}}"** pišu se promenljive, a blokovi su označeni sa **"{%"** i **"%}"**.

```
<html>
   <head>
      <title> Knjige </title>
   </head>
   <body>
      <h1> Knjige </h1>
      <ul>
         {% for knjiga in lista_knjiga %}
         <li>{{ knjiga.naziv }}</li>
         {% endfor %}
      </ul>
   </body>
</html>
```

U ovom bloku je realizovano da se za svaku knjigu, koja se nalazi u listi **lista_knjiga**, ispisuje njen naziv. Kao rezultat na stranici http://127.0.0.1:8000/najnovije/ će se pojaviti lista naziva knjiga poređana u opadajućem poretku po datumu objavljivanja. Ukoliko se obrasci smeštaju u posebni direktorijum, potrebno je to navesti u datoteci **settings.py**. Na primer, ukoliko se nalaze u direktorijumu **templates**, koji je u direktorijumu projekta, upisuje se:

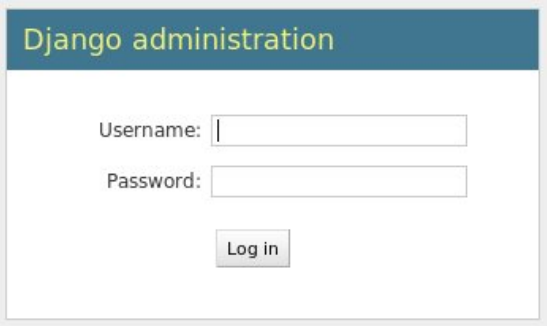
```
import os
PROJECT_PATH =  os.path.dirname(os.path.realpath(__file__))
TEMPLATE_DIRS = (
      os.path.join( PROJECT_PATH, 'templates' ),)
```

4.4 Administratorska stranica

Još jedna od pogodnosti koje Django okruženje pruža je kreiranje administratorske stranice na jednostavniji način. Korišćenje ove stranice je opciono i njeno pokretanje se sastoji iz nekoliko koraka:

- Na početak datoteke **models.py** potrebno je dodati **from django.contrib import admin**, a zatim za svaku klasu, koja treba da se prikaze, dodati **admin.site.register(ImeKlase)**.
- U datoteku **settings.py** u strukturu **INSTALLED_APPS** dodaje se **django.contrib.admin**.
- Vrši se sinhronizacija sa bazom komandom **python manage.py syncdb**.
- Kako bi se administratorska stranica pojavila na adresi **http://127.0.0.1:8000/admin/** , treba skloniti znak za komentar ispred **(r'^admin/', include('django.contrib.admin.urls'))** u datoteci **urls.py**.

Nakon ovih podeševanja, nakon pokretanja servera i ukucavanjem adrese administratorske stranice, u Internet pretraživaču pojaviće se stranica za prijavljivanje.



Slika 4: Strana za prijavljivanje

Unosi se korisničko ime i lozinka koji su podešeni prilikom dodavanja super korisnika kada je prvi put izvršena sinhronizacija sa bazom. Ukoliko je prijava uspešna, prikazuje se sledeća strana na kojoj administrator može da izabere opcije za uređivanje korisnika i grupe korisnika, a vidljive su i sve klase iz aplikacije u kojima postoji **admin.site.register(ImeKlase).**

Slika 5: Početna administratorska stranica

Klikom na veze *Add* i *Change*, moguće je dodavati podatke u bazu i menjati postojeće podatke. U primeru ove jednostavne aplikacije stranica za dodavanje sadržaja u bazu prikazana je na sledećoj slici:

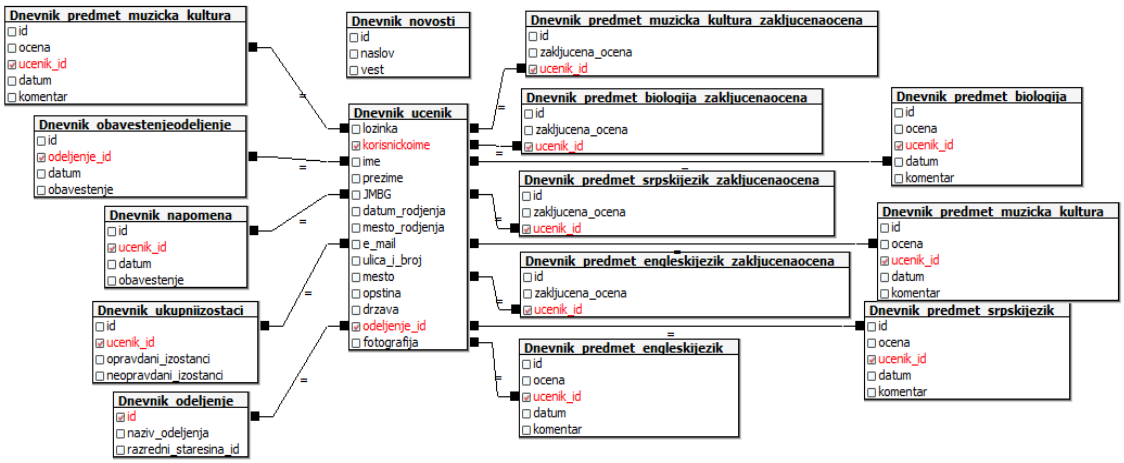
Slika 6: Forma za dodavanje podataka o knjizi

Preko administratorske stranice vodi se računa i o tome da li su podaci uneti u dobrom obliku i da li su popunjena polja koja su obavezna. Administrator dodaje korisnike koji će moći da koriste aplikaciju i određuje im pravila pristupa. Opcije za dodavanje, brisanje i izmene su vidljive samo korisnicima sa tim privilegijama. Postoji mnogo načina na koje korisnik može da prilagodi ovo okruženje svojim potrebama, a neki od njih su opisani u sledećem poglavlju na primeru aplikacije „*Elektronski dnevnik*“.

5. Aplikacija „Elektronski dnevnik“

Kao primer korišćenja programskog jezika Python u izradi Web aplikacija, napravljena je Web aplikacija koja omogućava praćenje ocena i izostanaka učenika putem Interneta. Takođe, olakšava komunikaciju između nastavnika jer će oni nakon prijave na aplikaciju moći da vide najnovije vesti o dešavanjima u školi i tako će uvek biti blagovremeno o tome obavešteni.

Kreirana je aplikacija i izvršena je sinhronizacija sa bazom podataka na način koji je opisan u prethodnom poglavlju. Korišćena je PostgreSQL baza, a bilo je potrebno instalirati i Python paket za rad sa ovom bazom pod nazivom PyGreSQL. Model baze podataka je osmišljen tako da se u što većoj meri iskoriste pogodnosti koje Django razvojno okruženje pruža. Na sledećoj slici je prikazan dijagram baze podataka, gde je svaki predmet izdvojen u posebnu tabelu kako bi predmetni nastavnik, koji se prijavi na aplikaciju, mogao da vidi i menja samo ocene iz predmeta koje on predaje. Radi preglednosti, na dijagramu nisu prikazani tabele za sve predmeta i zaključene ocene za te predmete.



Slika 7: Dijagram baze podataka

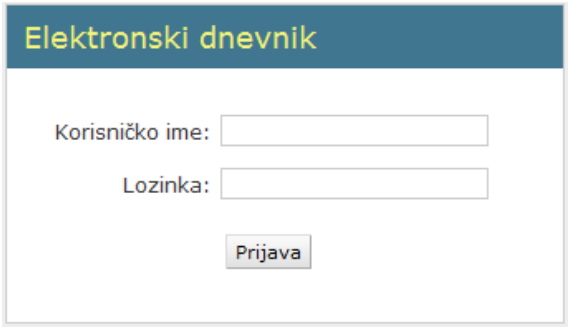
Sledećim delom koda dat je primer jedne Python klase koja se koristi za kreiranje modela i nalazi se u datoteci **models.py.**

```
class Ucenik(models.Model):
      korisnickoime = models.CharField("Korisničko ime",
max_length=50, primary_key = True)
      lozinka = models.CharField(max_length=20)
      ime = models.CharField(max_length=50)
      prezime = models.CharField(max_length=50)
      JMBG = models.CharField(max_length=13)
      datum_rodjenja = models.DateField("Datum rođenja")
      mesto_rodjenja = models.CharField("Mesto rođenja",
max_length=50)
      e_mail = models.EmailField("Elektronska pošta")
      ulica_i_broj = models.CharField(max_length=50)
      mesto = models.CharField(max_length=50)
```

```
opstina = models.CharField("Opština", max_length=50)
drzava = models.CharField("Država", max_length=50)
odeljenje = models.ForeignKey(Odeljenje)
fotografija = models.ImageField(upload_to="photo_images")
```

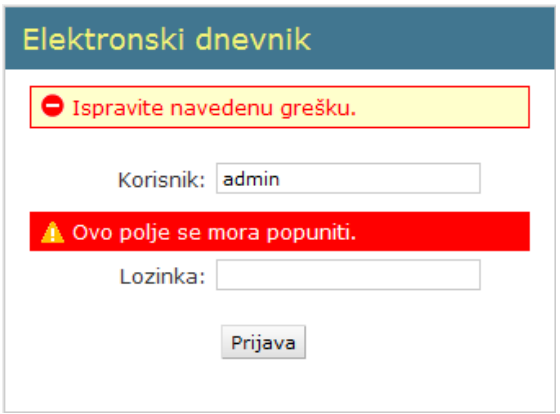
Na osnovu ove klase kreirana je tabela **Ucenik** sa kolonoma koje odgovaraju atributima klase. Tako kolona **korisnickoime** sadrži znakovne nizove maksimalne dužine 50 karaktera i predstavlja primarni ključ za ovu tabelu, dok atribut **odeljenje** definiše strani ključ na tabelu **Odeljenje**. Ukoliko se ne navede primarni ključ u modelu, Django kreira polje **id** koje je primarni ključ i sadrži celobrojnu vrednost koja se automatski uvećava za jedan. Fotografija se smešta u direktorijum **photo_images** jer je tako navedeno prilikom kreiranja modela.

Aplikacija ima više modula, koji su povezani sa različitim vrstama korisnika. Prilikom prve sinhronizacije sa bazom napravljen je korisnik sa administratorskim privilegijama čije je korisničko ime *„admin“* i lozinka *„admin“*. Kada se pokrene server na Internet adresi **http://127.0.0.1:8000/admin/** je stranica za prijavu.



Slika 8: Stranica za prijavu

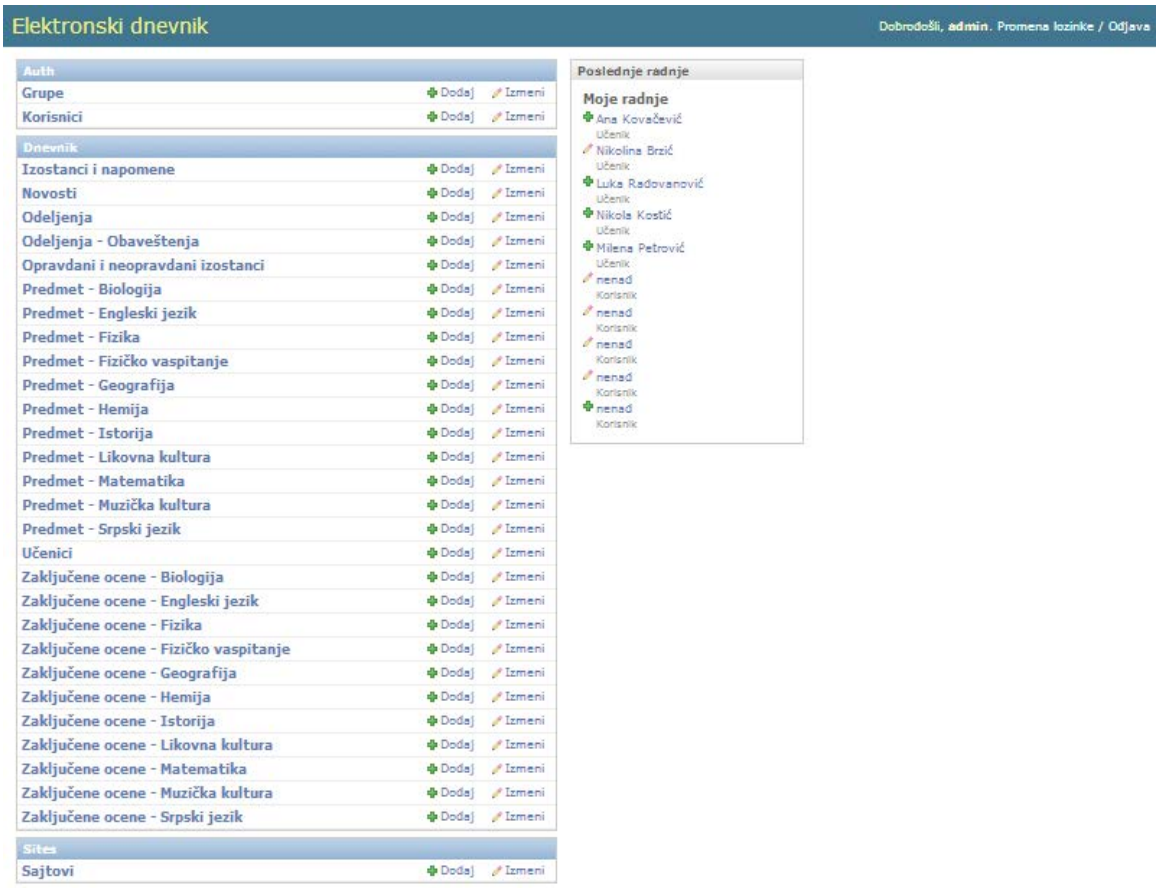
Promena izgleda stranice za prijavu se vrši kopiranjem Django datoteka **base_site.html** i **login.html** iz direktorijuma **Django-1.4\django\contrib\admin\templates\admin** u direktorijum **ElektronskiDnevnik\templates\admin**. Na primer, u ovoj aplikaciji je izvršena izmena da, umesto *„Django administration“*, na stranici za prijavu piše *„Elektronski dnevnik“*. Promena njihovog sadržaja će biti vidljiva prilikom pokretanja aplikacije. Django vodi računa o tome da li su uneta neophodna polja i upozorava na grešku.



Slika 9: Stranica za prijavu kada nije uneta lozinka

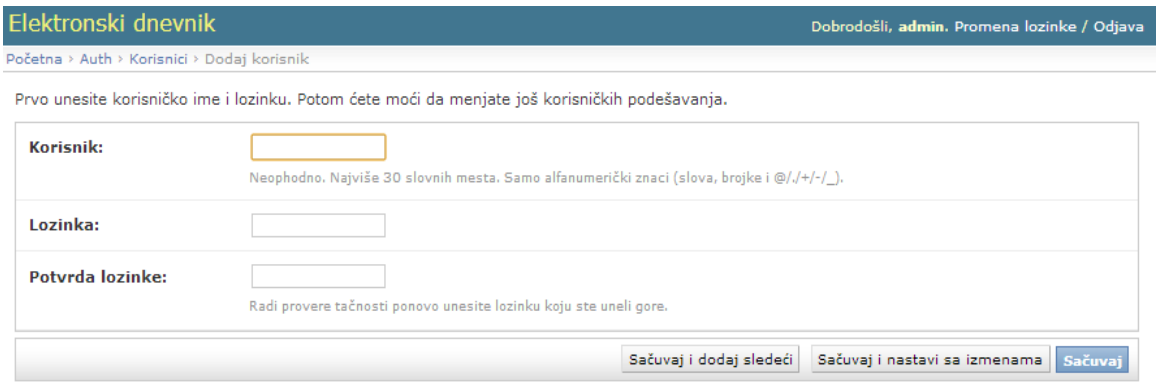
5.1 Stranica za administratora

Nakon uspešne prijave, administratoru su vidljive sve klase koje su definisane u datoteci **`models.py`**, a klikom na veze *Dodaj* ili *Izmeni*, unosi podatke u bazu i menja ih. Takođe, ima uvid u poslednje radnje koje je obavio prethodni put kada je bio prijavljen na aplikaciju.



Slika 10: Izgled administratorske stranice

Administrator dodaje nastavnike i dodeljuje im lozinke za pristup aplikaciji klikom na vezu *Dodaj*, koja se nalazi pored veze *Korisnici*.

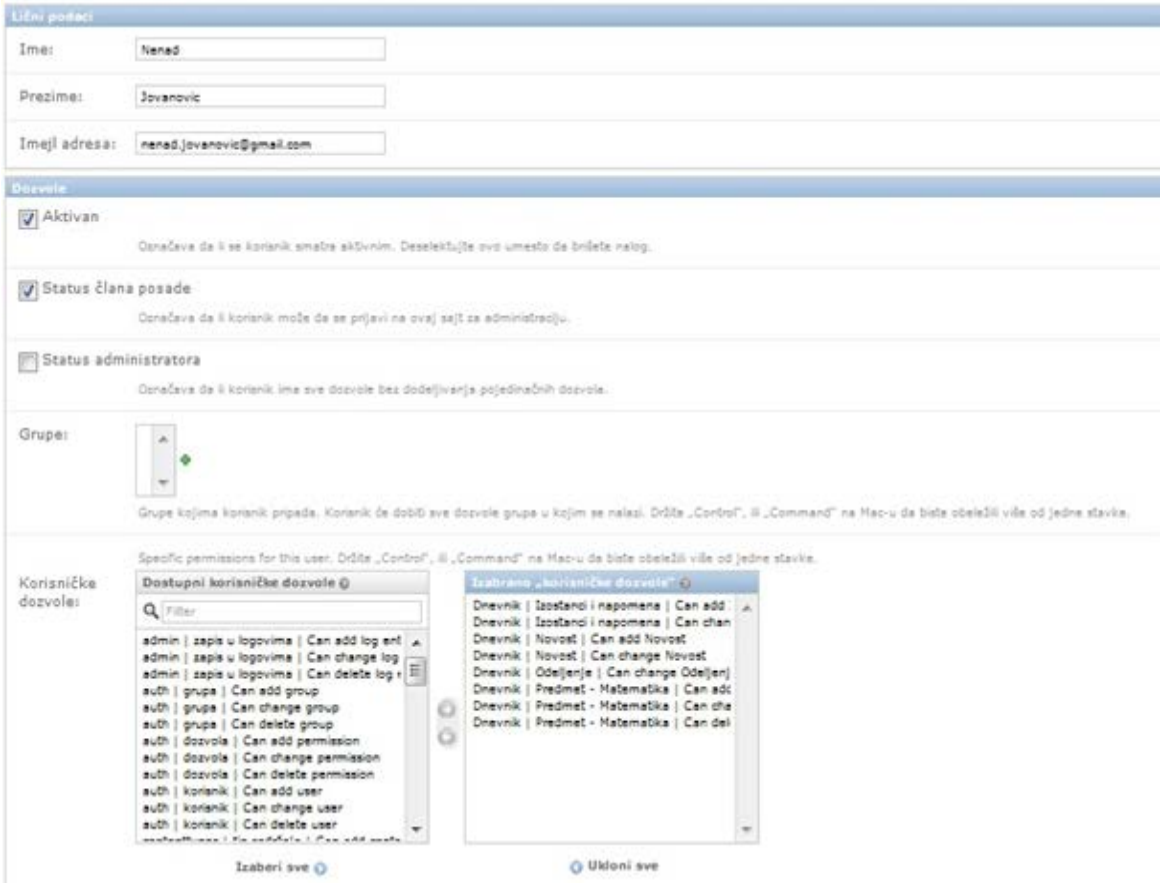


Slika 11: Stranica za dodavanje nastavnika

Nakon dodeljivanja lozinke, administrator unosi ime, prezime i imejl adresu nastavnika i dodeljuje mu osobine označavanjem polja:

- *Aktivan* – označava da li se korisnik smatra aktivnim. Ukoliko nastavnik bude prestao da radi u ovoj školi, nije potrebno brisati njegov nalog, već se on proglašava neaktivnim.
- *Status člana posade* – polje je označeno ako je korisniku omogućeno da se prijavi na sajt za administraciju i
- *Status administrator* – u ovoj aplikaciji ovo polje će uvek biti neoznačeno jer samo administrator ima sva prava.

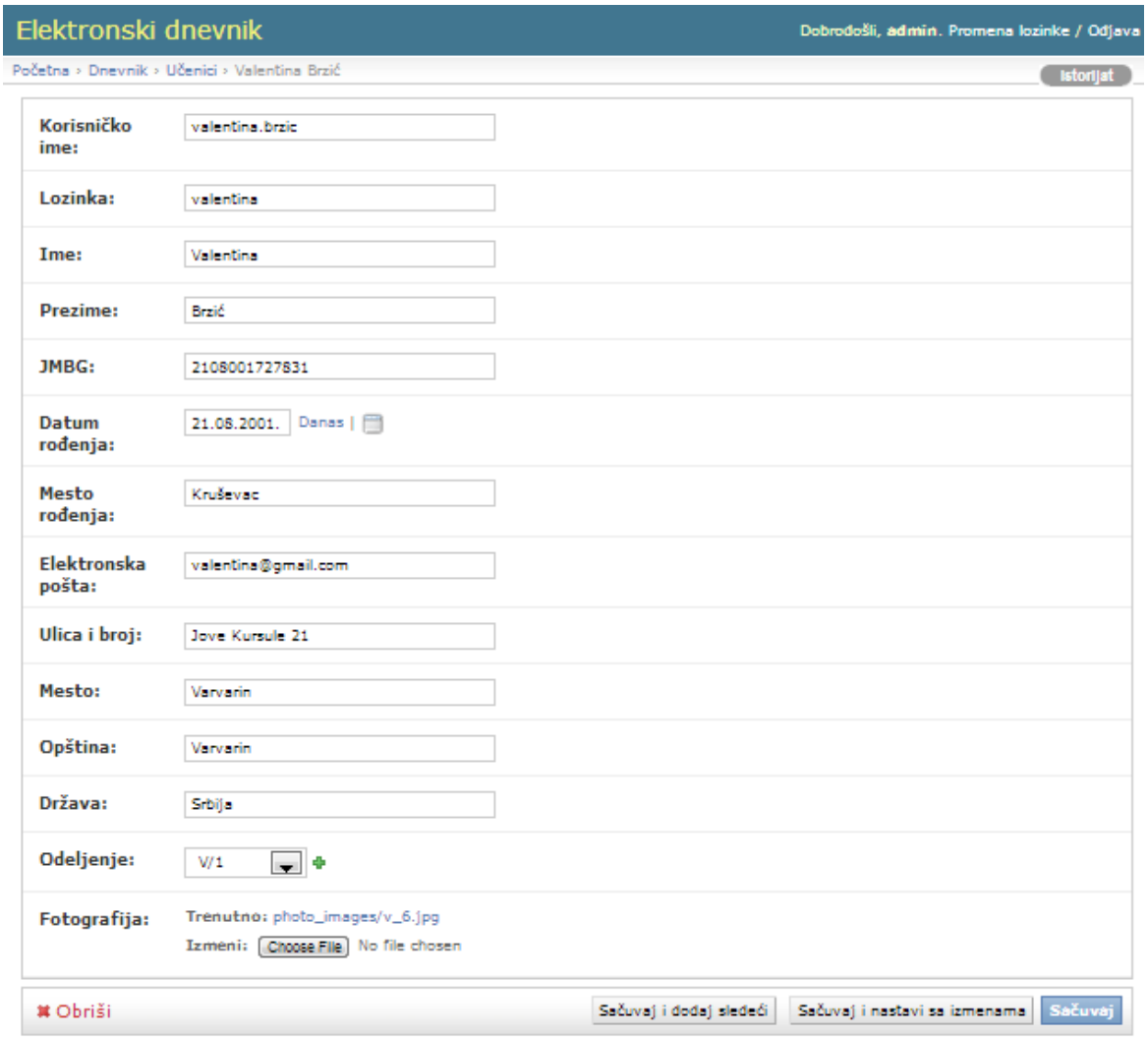
Moguće je korisnika svrstati u neku grupu, kao i to da se uz pomoć filtera precizira koja podatke iz baze može da menja. Nastavnik matematike će moći da menja samo ocene iz svog predmeta, dodaje izostanke učenicima, napomene za pojedinačnog učenika i na nivou odeljenja, kao i da dodaje novosti koje njegove kolege mogu da vide kada se prijave. Administratoru je omogućeno da vidi tačno vreme kada se nastavnik prijavljivao i koje podatke je menjao.



Slika 12: Podešavanja o nastavniku

Administrator dodaje podatke o učenicima klikom na vezu *Dodaj* pored polja *Učenici*. Svakom učeniku dodeljuje jedinstveno korisničko ime i lozinku za pristup aplikaciji. Zatim, unosi lične podatke učenika i klikom na dugme *Sačuvaj*, podaci će biti upisani u bazu, a administrator vraćen na početnu stranu. Ukoliko želi da odmah, nakon toga, nastavi sa unosom podataka o nekom drugom učeniku, može kliknuti na dugme

Sačuvaj i dodaj sledeći i prikazaće mu se stranica za unos podataka o učenicima sa praznim poljima za unos. Klikom na dugme *Sačuvaj i nastavi sa izmenama*, podaci će se upisati u bazu, bez prelaska na narednu stranu. Okruženje vrši provere da li su upisani podaci u sva obavezna polja, kao i da li su datum i e-mail ispravno uneti. Brisanje podataka o učeniku je omogućeno klikom na vezu *Obriši*.

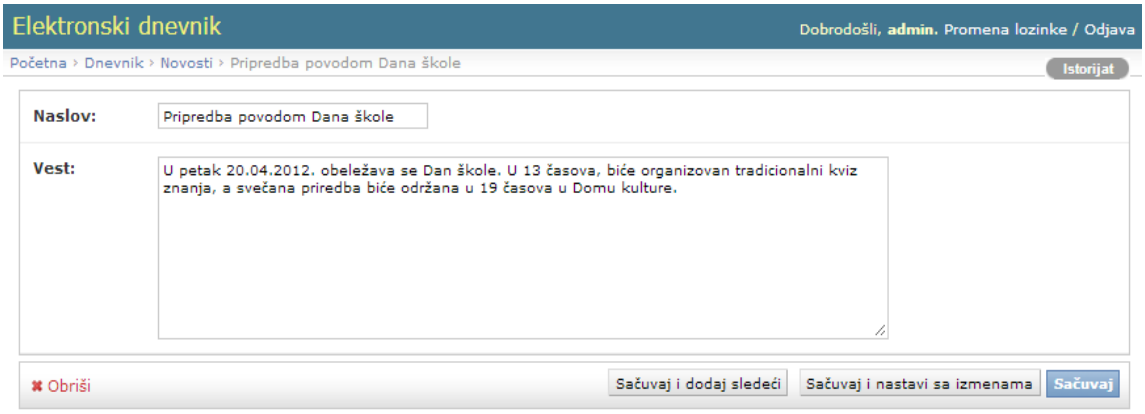


Slika 13: Stranica za dodavanje učenika

Klikom na vezu *Izmeni* pojavljuje se stranica sa spiskom svih učenika, odakle je moguće izabrati pojedinačne učenike i menjati podatke o njima.

Unos podataka o odeljenjima se vrši na sličan način kao što se unose podaci o učenicima.

Administrator, takođe, ima zadatak da objavljuje novosti vezane za dešavanja u školi, koje će moći da vide svi nastavnici kada se prijave na aplikaciju.



Slika 14: Stranica za dodavanje novosti

5.2 Stranica za nastavnike

Kada nastavnik pristupi aplikaciji ima na raspolaganju samo one opcije koje mu je administrator omogućio. Nastavnik matematike može da menja ocene iz matematike, upisuje izostanke učenicima i dodaje obaveštenje odeljenjima (što je vrlo korisno ukoliko, na primer, želi da obavesti sve učenike nekog odeljenja da čas neće biti održan). Takođe, ima mogućnost da pročita najnovije novosti o aktuelnostima u školi, ali i da dodaje nove. Tako, na primer, može da obavesti kolege o rezultatima sa takmičenja iz matematike, ili da pročita da dramska sekcija u školi organizuje predstavu. Na taj način je poboljšana komunikacija između nastavnika i olakšana njihova međusobna saradnja. Ukoliko je neki nastavnik razredni starešina odeljenja administrator će mu omogućiti pristup stranici za dodavanje opravdanih i neopravdanih izostanaka. Razredni starešina, takođe, može da dodaje učenike i menja njihove lične podatke.

Slika 15: Stranica za nastavnike

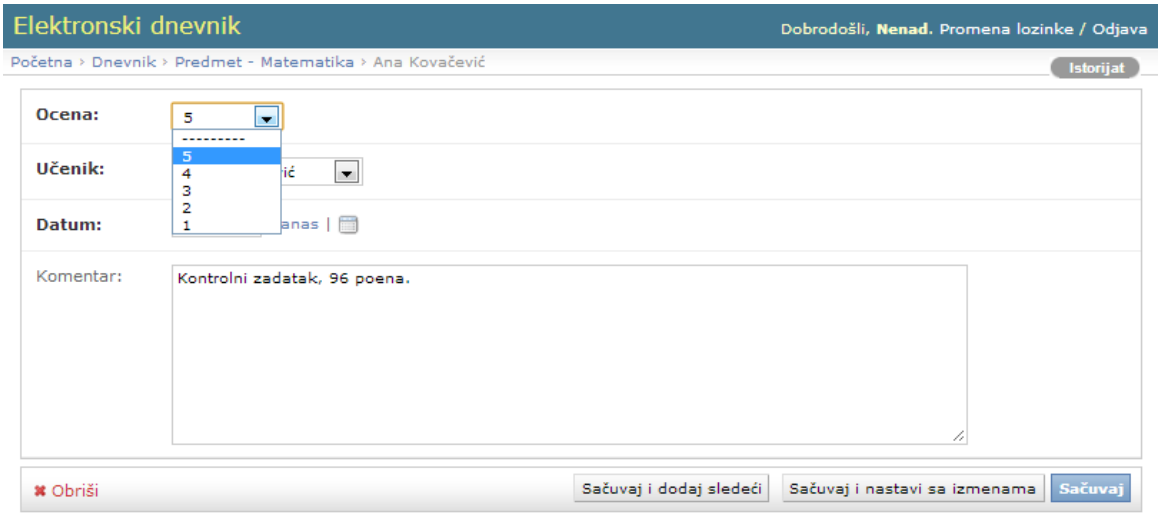
Kao i administrator, i nastavnik ima uvid u poslednje radnje koje je obavio kada je prethodni put bio prijavljen na aplikaciju. Ukoliko želi da proveri neke svoje izmene, može to uraditi klikom na određenu radnju. Otvara se stranica sa podacima koji su izmenjeni. Klikom na vezu *Istorijat*, koja se nalazi u gornjem desnom uglu, dolazi se do stranice sa tabelom, u kojoj za svaku promenu polja piše vreme kada se dogodila i koji korisnik ju je izvršio. Ako nastavnik zaključi da je neko nedozvoljeno menjao podatke, trebalo bi odmah o tome da obavesti administratora, koji će mu dodeliti novu šifru i vratiti stare podatke na osnovu kopije baze podataka. Preporučuje se da se svakodnevno pravi kopija podataka iz baze. Uvidom u istorijat izmena smanjena je mogućnost zloupotrebe i neovlašćenog menjanja ocena, što se sa tradicionalnim dnevnikom neretko dešavalo.

Slika 16: Stranica koja prikazuje istorijat izmena ocene iz matematike i datuma njenog unosa

Izborom opcije *Dodaj* pored naziva klase *Predmet – Matematika* otvara se stranica za unos nove ocene. Nastavniku je omogućeno da bira ocenu iz padajuće liste tako što je atributu **ocena** prosleđen parametar **choices = OCENE**, gde je **OCENE** prethodno definisana n-torka.

```
OCENE = ((5,'5'), (4,'4'), (3,'3'), (2,'2'), (1,'1'),)
ocena =  models.IntegerField(choices = OCENE)
```

Prvi član svake n-torke je celi broj koji se upisuje u bazu, a drugi član je niska koja se prikazuje u padajućoj listi. Atribut **datum** je tipa **DateField**, a klikom na polje *Danas* upisuje se današnji datum, kao datum unosa. Polje za komentar može ostati prazno, ali se preporučuje da nastavnik unese kratak opis ocene, na primer, da li je usmeno ili pismeno odgovaranje, ukoliko je bio kontrolni zadatak može upisati broj poena, i slično.



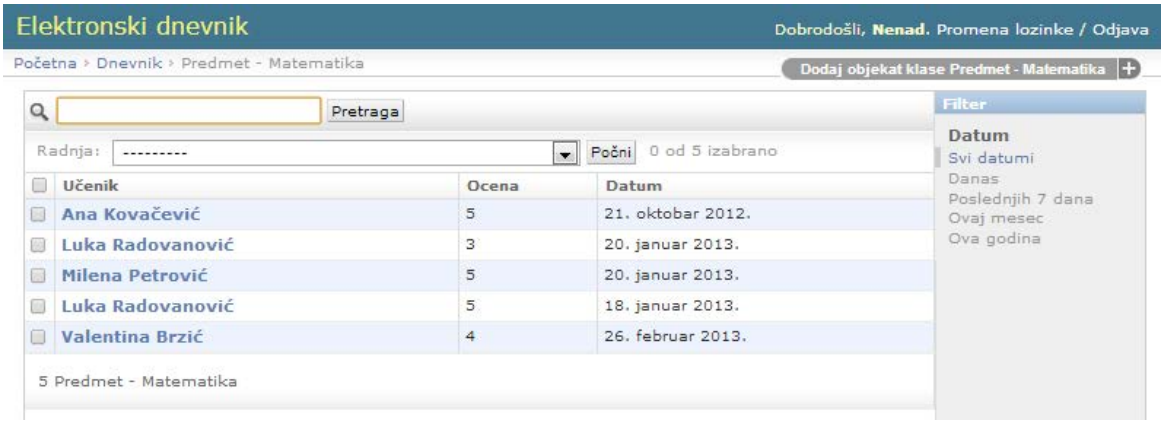
Slika 17: Stranica za upis ocena

Nakon izbora veze *Izmeni* pored naziva klase *Predmet - Matematika* sa početne strane, otvara se stranica sa listom svih objekata ove klase koji su uneti u bazu. Nastavnik iz liste bira učenika i zatim vrši izmene. Izgled stranice je prilagođen potrebama nastavnika definisanjem klasa koje nasleđuju klasu **ModelAdmin**. Klase se nalaze u datoteci **admin.py** i omogućavaju izbor kolona koje se prikazuju, polja po kojima se vrši pretraga i filtriranje, i slično.

```
from django.contrib import admin
from ElektronskiDnevnik.models import Predmet_Matematika
```

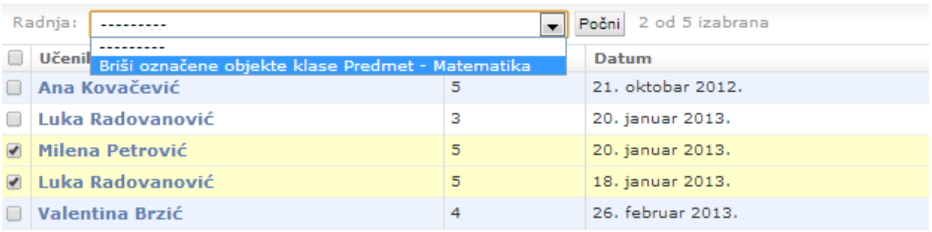
```
class Predmet_MatematikaAdmin(admin.ModelAdmin):
      list_display = ('ucenik', 'ocena', 'datum')
      search_fields = ['ucenik__ime', 'ucenik__prezime', 'datum',
'ocena']
      list_filter = ('datum',)
admin.site.register(Predmet_Matematika, Predmet_MatematikaAdmin)
```

Atributom **list_display** definisano je da u tabeli budu prikazane kolone sa imenima i prezimenima učenika, ocenama i datumima njihovog unosa. Atribut **search_fields** obezbeđuje da nastavnik može vršiti pretragu po svim poljima unosom određenog kriterijuma u polje za pretragu. Izborom određenog kriterijuma u delu *Filter*, koji se nalazi sa desne strane ekrana, sužava se lista učenika. Filtriranje se vrši po datumu unosa ocena, što je definisano atributom **list_filter**. Rezultati se sortiraju klikom na naslove kolona. Kako bi ove promene bile vidljive, neophodno je klasu **Predmet_MatematikaAdmin** pridružiti klasi **Predmet_Matematika** pozivom funkcije **admin.site.register()**, sa odgovarajućim argumentima.



Slika 18: Stranica sa listom unetih ocena

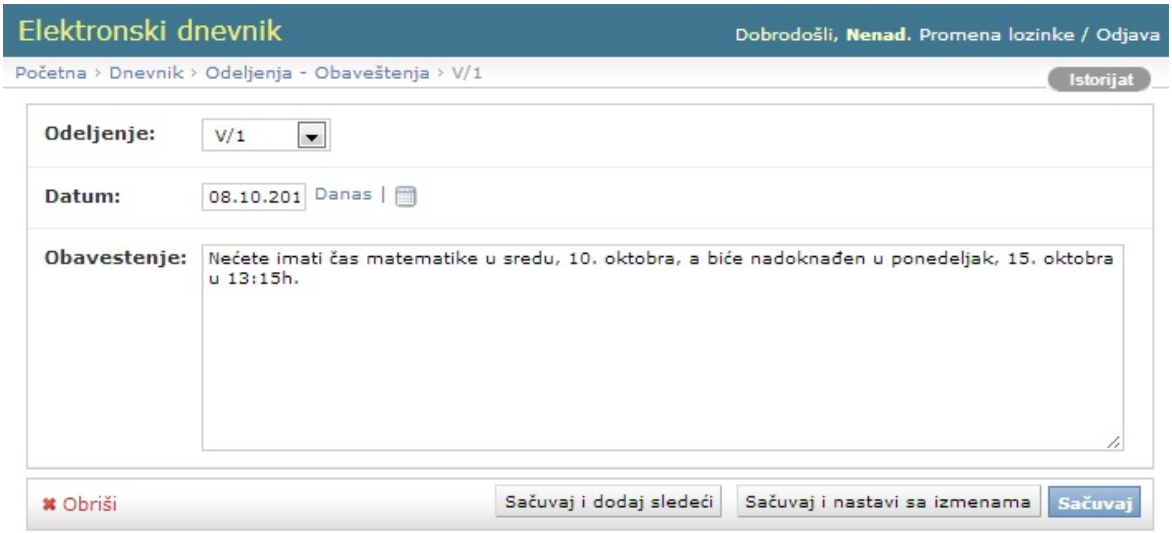
Moguće je obrisati više objekata odjednom, tako što se najpre označe objekti, zatim iz padajuće liste izabere opcija *Briši označene objekte klase Predmet – Matematika* i klikne na dugme *Počni*. Pojaviće se stranica sa sigurnosnim pitanjem da li nastavnik zaista želi da ukloni označene objekte.



Slika 19: Brisanje više objekata

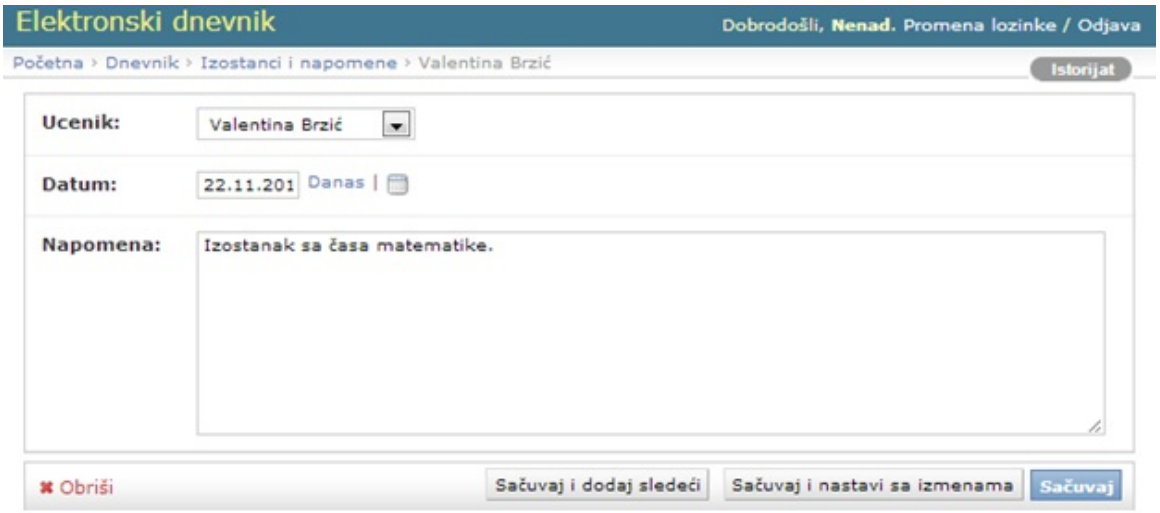
Na kraju polugodišta nastavnik unosi zaključene ocene izborom veze *Dodaj* pored naziva klase *Zaključena ocena – Matematika* sa početne strane. U padajućoj listi bira učenika i unosi njegovu zaključenu ocenu.

Ukoliko nastavnik želi da o nečemu obavesti sve učenike nekog odeljenja, potrebno je da izabere vezu *Odeljenja*, izabere odeljenje i u polje za obaveštenja unese željenu poruku. Poruka će biti vidljiva učenicima kada se prijave na aplikaciju.



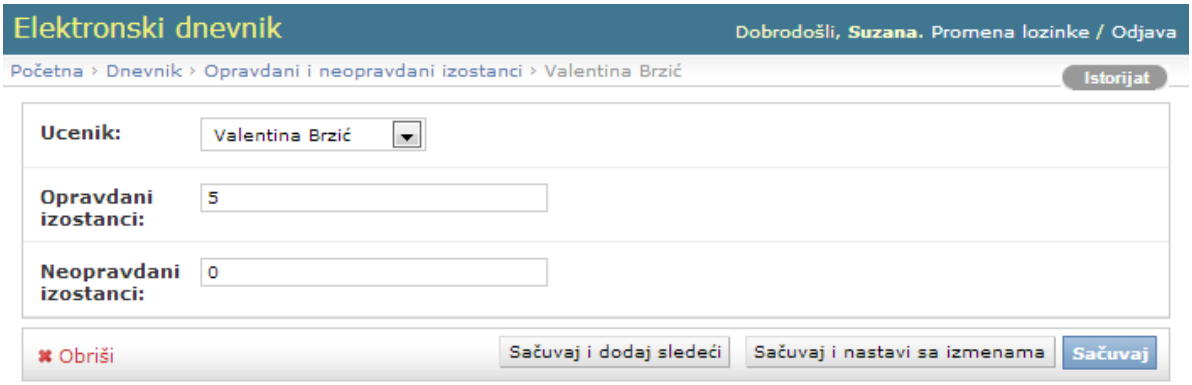
Slika 20: Stranica za upis poruke za sve učenike nekog odeljenja

Klikom na vezu *Izostanci i napomene*, nastavnik unosi izostanke za učenike, kao i napomenu ili obaveštenje za pojedinačnog učenika, što će moći da se pročita na učeničkoj stranici.



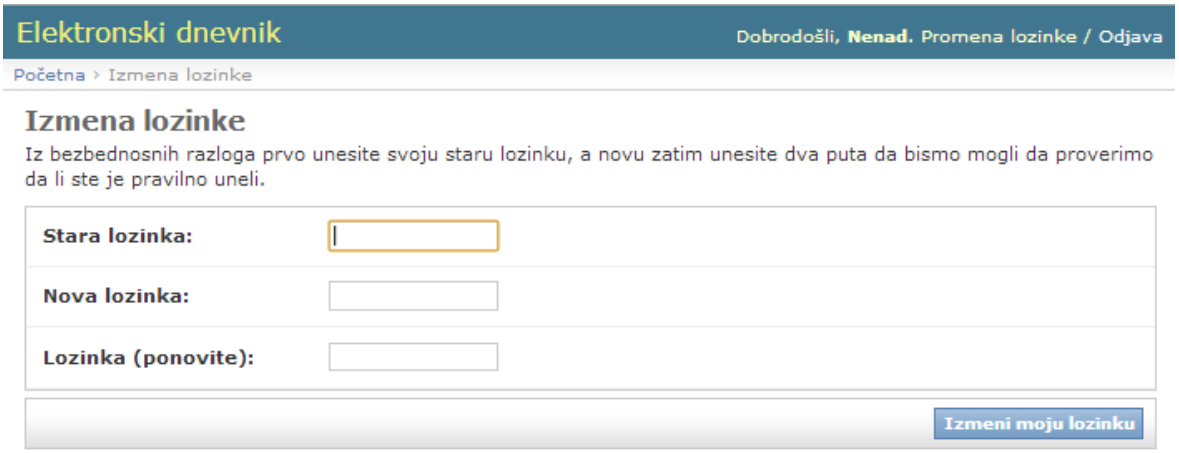
Slika 21: Stranica za upis izostanaka i napomena

Razredni starešina ima uvid u izostanke i napomene i, ukoliko je učenik dostavio opravdanje o sprečenosti da pohađa nastavu, upisuje opravdani izostak. Nakon odabira veze *Opravdani i neopravdani izostanci* otvara se stranica sa spiskom učenika i njihovih izostanaka. Razredni starešina može u polje za pretragu uneti ime i prezime učenika, a zatim, klikom na ime željenog učenika, prelazi se na stranicu za unos opravdanih i neopravdanih izostanaka.



Slika 22: Stranica za unos opravdanih i neopravdanih izostanaka

U gornjem desnom uglu ekrana nalazi se veza ka stranici za promenu lozinke, gde korisnik, radi bezbednosti, unosi staru lozinku i upisuje novu.



Slika 23: Stranica za promenu lozinke

5.3 Stranica za učenike i roditelje

Stranica sa koje učenici i roditelji pristupaju aplikaciji je na adresi http://localhost:8000/, što je definisano u datoteci **urls.py** dodavanjem reda **url(r'^$','ElektronskiDnevnik.views.prijava')** u promenljivu **urlpatterns**.

```
from django.conf.urls import patterns, url

urlpatterns = patterns('',
    # …
    url(r'^$','ElektronskiDnevnik.views.prijava'),
    # …
)
```

Pozivom stranice Django kreira objekat klase **HttpRequest**, koji sadrži podatke o zahtevu, i prosleđuje ga funkciji **prijava.** Objekat **request** ima dva atributa

request.GET i **request.POST**, koji sadrže podatke koje je korisnik uneo i imaju osobinu rečnika u Python-u. Funkcija **prijava** je definisana u datoteci **views.py** i poziva stranicu **strana_za_prijavu.html**, koja se nalazi u direktorijumu **templates**.

```
from django.http import HttpResponse
def prijava(request):
     return direct_to_template(request,'strana_za_prijavu.html',)
```

Datoteka **strana_za_prijavu.html** sadrži polja za upis korisničkog imena i lozinke, čije se vrednosti prosleđuju funkciji za pretragu učenika. Sledećim delom koda je predstavljena stranica za prijavu, odakle je, radi preglednosti, uklonjen deo vezan za način prikaza elemenata.

```
<html>
<head>
    <title> Prijava </title>
</head>
<body>
     <h1> Elektronski dnevnik </h1>
     Unesite korisničko ime i lozinku.
     <form action="/pretraga/" method="get">
          Korisnik:
          <input name="q" type="text"/>
          Lozinka:
          <input name="qr" type="password"/>
          <input type="submit" value="">
     </form>
</body>
</html>
```

Na prvoj stranici roditelji i učenici unose odgovarajuće podatke u polja za korisničko ime i lozinku, koje je administrator dodelio učeniku prilikom unosa podataka. Kroz promenljive **q** i **qr** metodom **get** prosleđuju se u URL **/pretraga/.**

Slika 24: Stranica za prijavu učenika za pristup aplikaciji

Promenljiva **urlpatterns** u datoteci **urls.py** sadrži i **url(r'^pretraga/$', 'ElektronskiDnevnik.views.pretraga')**, usled čega se poziva funkcija **pretraga**. Uz pomoć objekta **request.GET** moguć je pristup vrednostima **q** i **qr**, jer se ovaj objekat ponaša kao rečnik, gde se podaci čuvaju u obliku parova ključ:vrednost. Pre pretrage vrši se provera da li se **'q'** i **'qr'** nalaze u objektu, kao i da li je njihova vrednost različita od prazne niske. Na osnovu dobijenih vrednosti traži se objekat klase **Ucenik**, sa odgovarajućim korisničkim imenom i lozinkom. Takođe, za sve predmete izdvaju se objekti sa odgovarajućim korisničkim imenom učenika, kako bi na stranici bile prikazane sve ocene.

```
def pretraga(request):
      if 'q' in request.GET and request.GET['q'] and 'qr' in
      request.GET and request.GET['qr']:
            q = request.GET['q']
            qr = request.GET['qr']
           uc = Ucenik.objects.filter(korisnickoime=q).
           filter(lozinka=qr)
            sj = Predmet_SrpskiJezik.objects.filter(ucenik_id=q).
            order_by('-datum')
```

Ovde je naveden samo deo funkcije, a u celosti se može naći u dodatku. Izdvajaju se i sve zaključene ocene i, na osnovu njih, računa prosečna ocena. Takođe, izdvojeni su i izostanci i obaveštenja. Poziva se funkcija **direct_to_template**, koja uzima podatke i prosleđuje ih stranici za prikaz podataka o učenicima. Prosleđuju se objekti i pristupa se njihovim atributima. Narednim delom koda kreira se tabela sa ocenama iz srpskog jezika i datumima njihovog unosa.

```
<table>
      <tr>
            <th> Datum unosa </th>
            <th> Ocene </th>
      </tr>
      {% for srpskiJezik in sj %}
      <tr>
            <td> {{ srpskiJezik.datum }} </td>
            <td rel="tooltip" title="{{ srpskiJezik.komentar }}">
            {{ srpskiJezik.ocena }} </td>
      </tr>
      {% endfor %}
</table>
```

Slično je i za vrednosti u ostalim tabelama.

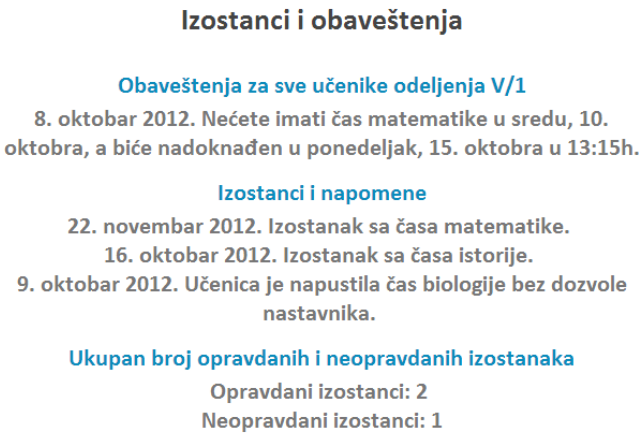
Nakon uspešne prijave, pojaviće se stranica sa svim ocenama, izostancima i obaveštenjima. U gornjem delu stanice nalaze se lični podaci učenika.

Slika 25: Deo stranice sa podacima i ocenama učenika

Ocene su grupisane po predmetima i pored svake od njih piše i datum kada je uneta u elektronski dnevnik. Ukoliko roditelji žele da vide komentar, koji je nastavnik uneo prilikom upisa ocene, potrebno je da prevuku mišem preko ocene i pojaviće se podaci ispod kursora miša. Na stranici su ispisane sve unete zaključene ocene i njihov prosek.

Slika 26: Detalji unete ocene

Klikom na vezu *Izostanci i obavešenja*, prikazuju se izostanci koje je učenik napravio, kao i obaveštenja koja su nastavnici uneli. To mogu biti obaveštenja za sve učenike odeljenja, kao i napomene za pojedinačne učenike. Ukoliko je učenik napravio neki prekršaj, nastavnik je uneo opis prekršaja i datum i to će roditelji videti u odeljku sa izostancima i napomenama. Kod izostanaka se navodi predmet i datum, kada je učenik izostao sa časa.



Slika 27: Prikaz izostanaka i obaveštenja

Roditelji mogu odštampati ocene i izostanke svog deteta klikom na vezu *Štampa*, koja se nalazi u gornjem desnom uglu. Klikom na vezu *Odjava*, korisnik se preusmerava na početnu stranu.

6. Zaključak

Ovim radom je prikazan način izrade Web aplikacije korišćenjem Django razvojnog okruženja, koje se zasniva na programskom jeziku Python. Aplikacija je veoma fleksibilna i može se lako nadograđivati i poboljšati, u zavisnosti od potreba škola. Prednost joj je i to što se može izvršavati na različitim operativnim sistemima i koristi tehnologije koje su besplatne, tako da škole neće morati da izdvajaju velike količine novca za skupoceni softver. Pored toga što ova aplikacija pruža roditeljima uvid u školske aktivnosti dece i olakšava komunikaciju između roditelja i nastavnika, smanjena je i mogućnost zloupotrebe dnevnika. Neretko se dešavalo da učenici ukradu dnevnik i neovlašćeno ispravljaju i dopisuju ocene, što je sa elektronskim dnevnikom nemoguće. Unete ocene u računar se arhiviraju i čuvaju generacijama u više digitalnih kopija. Takođe, učenici često kriju neopravdane izostanke i loše ocene mesecima plašeći se reakcije roditelja, ali upotrebom ove aplikacije roditelji će biti obavešteni, moći će da pronađu uzroke neuspeha i blagovremeno pruže pomoć svom detetu. Tako je približavanje škole roditeljima veoma značajno i u vaspitanju dece i dovodi do poboljšanja uspeha učenika.

7. Dodatak

```
def pretraga(request):
    if 'q' in request.GET and request.GET['q'] and 'qr' in
request.GET and request.GET['qr']:
        q = request.GET['q']
        qr = request.GET['qr']
        uc =
Ucenik.objects.filter(korisnickoime=q).filter(lozinka=qr)
        sj =
Predmet_SrpskiJezik.objects.filter(ucenik_id=q).order_by('-
datum')
        sjzo =
Predmet_SrpskiJezik_ZakljucenaOcena.objects.filter(ucenik_id=q)
        mat =
Predmet_Matematika.objects.filter(ucenik_id=q).order_by('-datum')
        matzo =
Predmet_Matematika_ZakljucenaOcena.objects.filter(ucenik_id=q)
        ej =
Predmet_EngleskiJezik.objects.filter(ucenik_id=q).order_by('-
datum')
        ejzo =
Predmet_EngleskiJezik_ZakljucenaOcena.objects.filter(ucenik_id=q)
        f =
Predmet_Fizika.objects.filter(ucenik_id=q).order_by('-datum')
        fzo =
Predmet_Fizika_ZakljucenaOcena.objects.filter(ucenik_id=q)
        h =
Predmet_Hemija.objects.filter(ucenik_id=q).order_by('-datum')
        hzo =
Predmet_Hemija_ZakljucenaOcena.objects.filter(ucenik_id=q)
        b =
Predmet_Biologija.objects.filter(ucenik_id=q).order_by('-datum')
        bzo =
Predmet_Biologija_ZakljucenaOcena.objects.filter(ucenik_id=q)
        g =
Predmet_Geografija.objects.filter(ucenik_id=q).order_by('-datum')
        gzo =
Predmet_Geografija_ZakljucenaOcena.objects.filter(ucenik_id=q)
        i =
Predmet_Istorija.objects.filter(ucenik_id=q).order_by('-datum')
        izo =
Predmet_Istorija_ZakljucenaOcena.objects.filter(ucenik_id=q)
        mk =
Predmet_Muzicka_kultura.objects.filter(ucenik_id=q).order_by('-
datum')
        mkzo =
Predmet_Muzicka_kultura_ZakljucenaOcena.objects.filter(ucenik_id=
q)
        lk =
Predmet_Likovna_kultura.objects.filter(ucenik_id=q).order_by('-
datum')
```

```
            likzo = P
redmet_Likovna_kultura_ZakljucenaOcena.objects.filter(ucenik_id=q
)
            fv =
Predmet_Fizicko_vaspitanje.objects.filter(ucenik_id=q).order_by('
-datum')
            fizzo =
Predmet_Fizicko_vaspitanje_ZakljucenaOcena.objects.filter(ucenik_
id=q)
            nap = Napomena.objects.filter(ucenik_id=q).order_by('-
datum')
            ui = UkupniIzostaci.objects.filter(ucenik_id=q)
            o = ObavestenjeOdeljenje.objects.all().order_by('-
datum')
            rez1 = 0
            brojac=0

            for mzo in matzo:
                  rez1 = rez1 + mzo.zakljucena_ocena
                  if mzo.zakljucena_ocena!="":
                        brojac+=1

            for szo in sjzo:
                  rez1 = rez1 + szo.zakljucena_ocena
                  if szo.zakljucena_ocena!="":
                        brojac+=1

            for ezo in ejzo:
                  rez1 = rez1 + ezo.zakljucena_ocena
                  if ezo.zakljucena_ocena!="":
                        brojac+=1

            for fi in fzo:
                  rez1 = rez1 + fi.zakljucena_ocena
                  if fi.zakljucena_ocena!="":
                        brojac+=1

            for he in hzo:
                  rez1 = rez1 + he.zakljucena_ocena
                  if he.zakljucena_ocena!="":
                        brojac+=1

            for bi in bzo:
                  rez1 = rez1 + bi.zakljucena_ocena
                  if bi.zakljucena_ocena!="":
                        brojac+=1

            for ge in gzo:
                  rez1 = rez1 + ge.zakljucena_ocena
                  if ge.zakljucena_ocena!="":
                        brojac+=1

            for isto in izo:
                  rez1 = rez1 + isto.zakljucena_ocena
                  if isto.zakljucena_ocena!="":
                        brojac+=1

            for mu in mkzo:
                  rez1 = rez1 + mu.zakljucena_ocena
                  if mu.zakljucena_ocena!="":
```

```
                    brojac+=1
               for li in likzo:
                    rez1 = rez1 + li.zakljucena_ocena
                    if li.zakljucena_ocena!="":
                         brojac+=1
               for fiv in fizzo:
                    rez1 = rez1 + fiv.zakljucena_ocena
                    if fiv.zakljucena_ocena!="":
                    brojac+=1
               if brojac == 0:
                    prosek=0
               else:
                    prosek = rez1/float(brojac)

               prosek = round(prosek,2)

              return
direct_to_template(request,'rezultat_pretrage.html',{'matzo':matz
o,'ui':ui,'sjzo':sjzo,'mat':mat,'o':o,'sj':sj,'ej':ej,'ejzo':ejzo
,'f':f,'fzo':fzo,'h':h,'hzo':hzo,'b':b,'bzo':bzo,'g':g,'gzo':gzo,
'i':i,'izo':izo,'mk':mk,'mkzo':mkzo,'lk':lk,'likzo':likzo,'fv':fv
,'fizzo':fizzo,'uc':uc,'rez1':rez1,'prosek':prosek,'nap':nap})
          else:
               return HttpResponse('Popunite sva polja!')
```

8. Literatura

[1] Majkl Doson, ***Python: uvod u programiranje,*** Mikroknjiga, Beograd, 2010.

[2] http://www.python.org/psf/

[3] http://sr.wikipedia.org/sr/Пајтон_(програмски_језик)

[4] http://www.python.org/

[5] Wesley J. Chun, ***Core Python Programming,*** Prentice Hall PTR, Upper Saddle River, NJ, 2001.

[6] Jennifer Campbell, Paul Gries, Jason Montojo, Greg Wilson, ***Practical Programming: An Introduction to Computer Science Using Python,*** Pragmatic Bookshelf, USA, 2009.

[7] Swaroop C.H., ***A Byte of Python,*** http://www.swaroopch.com, 2008.

[8] Adrian Holovaty, Jacob Kaplan-Moss, ***The Django Book,*** Apress, 2007.

[9] http://en.wikipedia.org/wiki/Django_(web_framework)

[10] https://www.djangoproject.com/

[11] Jeff Forcier, Paul Bissex, Wesley Chun, Python Web Development with Django, Addison Wesley, USA, 2009.