Primjer objektno-orijentirane baze podataka u sustavu ZODB

Krajačić, Petar

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Petar Krajačić

Primjer objektno-orijentirane baze podataka u sustavu ZODB

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Petar Krajačić

JMBAG: 0016123652

Studij: Primjena informacijske tehnologija u poslovanju

PRIMJER OBJEKTNO-ORIJENTIRANE BAZE PODATAKA U SUSTAVU ZODB

ZAVRŠNI RAD

Mentor:

Izv. prof. dr. sc. Markus Schatten

Varaždin, rujan 2023

Petar Krajačić

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sadržaj

1. Uvod

U suvremenom digitalnom dobu, organizacija osobnih i profesionalnih zadataka postaje sve važnija kako bi se postigla produktivnost i učinkovitost. U tom kontekstu, aplikacije za vođenje popisa zadataka, poznate kao To-do liste, igraju ključnu ulogu u olakšavanju upravljanja zadacima i obavezama. Razvoj takvih aplikacija zahtijeva pažljivo razmišljanje o modeliranju podataka i njihovoj učinkovitoj pohrani kako bi se osigurala pouzdana i brza izvedba.

Ovaj završni rad istražuje primjenu objektno-orijentirane baze podataka u kontekstu razvoja To-do liste i pruža praktičan primjer upotrebe sustava ZODB (Zope Object Database) za pohranu i upravljanje To-do stavkama. Objektno-orijentirane baze podataka donose mnoge prednosti u smislu organizacije podataka i njihove prilagodljivosti, što ih čini atraktivnim izborom za razvoj aplikacija poput To-do lista.

Aplikacija To-do lista koja će biti predstavljena u ovom radu omogućuje korisnicima jednostavno dodavanje, uređivanje i brisanje To-do stavki, označavanje njihove važnosti te postavljanje rokova za zadatke. Kroz ovu aplikaciju, istražit ćemo kako ZODB, kao objektno-orijentirana baza podataka, olakšava pohranu i dohvat podataka o To-do stavkama te kako se njegova upotreba može primijeniti na stvarne scenarije upravljanja zadacima.

Ovaj rad će se temeljiti na konkretnom primjeru implementacije To-do liste koristeći ZODB, prateći proces modeliranja podataka, pohranjivanja i upravljanja To-do stavkama. Također će razmotriti prednosti i izazove koje donosi korištenje objektno-orijentirane baze podataka u kontekstu aplikacija za upravljanje zadacima.

Kroz ovu analizu, cilj ovog rada je pružiti uvid u primjenu objektno-orijentirane baze podataka u stvarnim aplikacijama te ilustrirati kako se takve tehnike mogu koristiti za rješavanje specifičnih problema u području razvoja softvera.

2. Pregled literature

U ovom odjeljku pružiti ću pregled relevantne literature i teorijskih postavki u vezi s objektno-orijentiranim bazama podataka (OOB) i konkretnom implementacijom, ZODB (Zope Object Database). Pregled literature pomoći će u razumijevanju osnovnih koncepta i prednosti OOB-a, kao i pružiti uvid u ZODB kao objektno-orijentiranu bazu podataka.

2.1. Objektno-orijentirane baze podataka

Objektno-orijentirane baze podataka (OOB) predstavljaju evoluciju koncepta relacijskih baza podataka. OOB pristup podrazumijeva pohranu podataka u obliku objekata, čime se omogućuje modeliranje stvarnog svijeta unutar baze podataka. Osnovne karakteristike OOB-a uključuju:

- **Objekti kao temeljne jedinice**: U OOB-u, podaci se pohranjuju u obliku objekata koji sadrže svoje atribute i metode za manipulaciju tim podacima.
- **Složene strukture podataka**: OOB-i omogućuju pohranu i upravljanje složenim strukturama podataka kao što su liste, stabla i grafovi.
- **Očuvanje integriteta podataka**: OOB-i imaju ugrađene mehanizme za očuvanje integriteta podataka, što olakšava razvoj i održavanje aplikacija [1].

Iako OOB-i nude niz prednosti, uključujući čitljiviji kod i olakšano modeliranje stvarnog svijeta, također postoje izazovi kao što su performanse i kompatibilnost s postojećim sustavima [2].

2.1.1. Osnovni koncepti

2.1.1.1. Klasa kao osnovna struktura

Klasa predstavlja osnovni konstrukcijski element u objektno-orijentiranom modeliranju. To je apstraktan koncept koji definira skup atributa (članova) i metoda (operacija) koje će objekti te klase imati. Klasa služi kao šablona za stvaranje pojedinačnih objekata.

2.1.1.2. Slogovi kao instance klasa

Slogovi su pojedinačni objekti koji pripadaju određenoj klasi. Svaki slog ima svoj vlastiti skup vrijednosti atributa, ali dijeli iste metode definirane u klasi. Slogovi su konkretni reprezentanti apstraktne klase.

2.1.1.3. Identitet objekta kao jedinstveni identifikator

Identitet objekta je jedinstveni identifikator koji svakom objektu u bazi podataka dodjeljuje jedinstvenu vrijednost. To omogućava razlikovanje i referenciranje pojedinačnih objekata u bazi. Identitet objekta često se koristi kao ključ za brzu pretragu i pristup objektima.

2.1.1.4. Operator pristupa za pristup članovima objekta

Operator pristupa (često se koristi točka, ".") omogućava pristup članovima objekta, uključujući atribute i metode. Ovo je osnovni mehanizam za interakciju s objektima i manipulaciju njihovim podacima.[3]

2.1.2.Funkcije i Metode

2.1.2.1. Razlika između funkcija i metoda

Funkcije i metode su oba načina obrade podataka unutar OOOBP. Funkcije su obično globalne i neovisne o konkretnim objektima, dok su metode vezane uz određeni objekt i imaju pristup članovima tog objekta. Metode su ključne za implementaciju ponašanja objekata.

2.1.2.2. Upotreba identiteta objekta u funkcijama i metodama

Identitet objekta često se koristi kao argument u funkcijama i metodama kako bi se identificirao specifičan objekt koji treba obraditi. Ovo omogućava manipulaciju podacima tog objekta.[3]

2.1.3. Učahurenje

2.1.3.1. Važnost učahurenja

Učahurenje, ili enkapsulacija, predstavlja ključni koncept u OOOBP. Ono omogućava pristup podacima objekta bez obzira na način njihove reprezentacije u

konkretnoj strukturi podataka. Pristup podacima obavlja se putem metoda, što osigurava integritet podataka i neovisnost o unutarnjoj strukturi objekta.

2.1.3.2. Sučelja za pristup podacima

Sučelja su definicije metoda koje objekti moraju implementirati kako bi podržavali određenu funkcionalnost. Sučelja omogućavaju programerima da pristupaju objektima bez potrebe za poznavanjem njihove unutarnje strukture.[3]

2.1.4. Polimorfizam

2.1.4.1. Pojam polimorfizma u OOOBP

Polimorfizam se odnosi na sposobnost različitih objekata da reagiraju na iste metode ili funkcije na različite načine. To omogućava dinamičku promjenu ponašanja objekata ovisno o kontekstu ili konkretnim instancama.

2.1.4.2. Razlike u arnosti i domenama funkcija i metoda

Polimorfizam se često postiže varijacijama u arnosti (broju argumenata) ili domenama (tipovima argumenata) funkcija i metoda. Ovo omogućava istim metodama ili funkcijama da rade s različitim vrstama podataka ili brojem argumenata.[3]

2.1.5. Nasljeđivanje

2.1.5.1. Koncept nasljeđivanja u objektno-orijentiranom modeliranju

Nasljeđivanje omogućava da pojedine klase nasljeđuju strukturu i ponašanje od drugih klasa. Klasa koja nasljeđuje naziva se dijete, dok je klasa koja se nasljeđuje roditelj. Nasljeđivanje omogućava ponovno korištenje koda, organizaciju klasa u hijerarhiju i dodavanje specifičnih atributa ili metoda u naslijeđene klase.

2.1.5.2. Višestruko nasljeđivanje i njegova primjena

Višestruko nasljeđivanje omogućava da jedna klasa nasljeđuje od više različitih klasa. Ovo može povećati složenost, ali također omogućava veću fleksibilnost i ponovnu upotrebu koda. Primjene višestrukog nasljeđivanja treba pažljivo razmotriti kako bi se održala jasna hijerarhija klasa i izbjegle konflikte.[3]

2.2. Zope Object Database (ZODB)

Zope Object Database (ZODB) predstavlja konkretnu implementaciju objektno-orijentirane baze podataka koja je posebno popularna u svijetu Python programskog jezika. Ključne značajke ZODB-a uključuju:

- **Pohrana Python objekata**: ZODB pohranjuje Python objekte i njihove međusobne veze na način koji je sličan radu Python interpretera. Ovo omogućuje programerima da rukuju objektima na prirodan način, bez potrebe za konverzijom podataka između različitih formata.
- **Transakcije**: Baza podržava transakcije, što omogućuje konzistentno čitanje i pisanje podataka. Transakcije osiguravaju da promjene u bazi budu atomične i konzistentne.
- **Verzioniranje objekata**: ZODB omogućuje verzioniranje objekata, što je korisno za praćenje promjena i povijesti podataka. Ovo je posebno važno u aplikacijama koje zahtijevaju povijest promjena, poput sustava za upravljanje dokumentima.
- **Indeksiranje podataka**: Za brzi dohvat podataka, ZODB podržava indeksiranje podataka. To omogućuje brz i efikasan pristup podacima, čak i u velikim bazama [4].

Razumijevanje ZODB-a pomoći će nam bolje razumjeti kako se koncepti objektno-orijentiranih baza podataka primjenjuju u stvarnim aplikacijama, posebno u kontekstu Python programskog jezika.

2.3. Korišteni alati i tehnologije

Pri izradi primjera aplikacije koristio sam niz alata i tehnologija, uključujući:

- **Python**: Python je programski jezik u kojem je razvijen ZODB i koji se koristi za razvoj aplikacije.[5]
- **Tkinter**: Tkinter je standardna Python biblioteka za izradu grafičkog korisničkog sučelja (GUI), a koristi se za stvaranje korisničkog sučelja naše aplikacije.[6]

- **ZODB**: ZODB je centralna tehnologija koja omogućuje pohranu i upravljanje objektima u aplikaciji.
- **FileStorage**: FileStorage je dio ZODB-a koji se koristi za pohranu objekata u datoteke na disku.
- **Tkinter biblioteke za dijaloge i poruke**: Koristili smo Tkinterove biblioteke poput simpledialog i messagebox za interakciju s korisnikom kroz dijaloge i prikazivanje poruka.

3. Implementacija

U ovom odjeljku, detaljno ću opisati strukturu i implementaciju aplikacije za upravljanje zadacima koristeći objektno-orijentiranu bazu podataka ZODB. Prikazati ću kako su koncepti OOB-a primijenjeni u stvarnom projektu, uključujući opis arhitekture aplikacije, dijagram klasa, ključne dijelove programskog koda i postavke sustava.

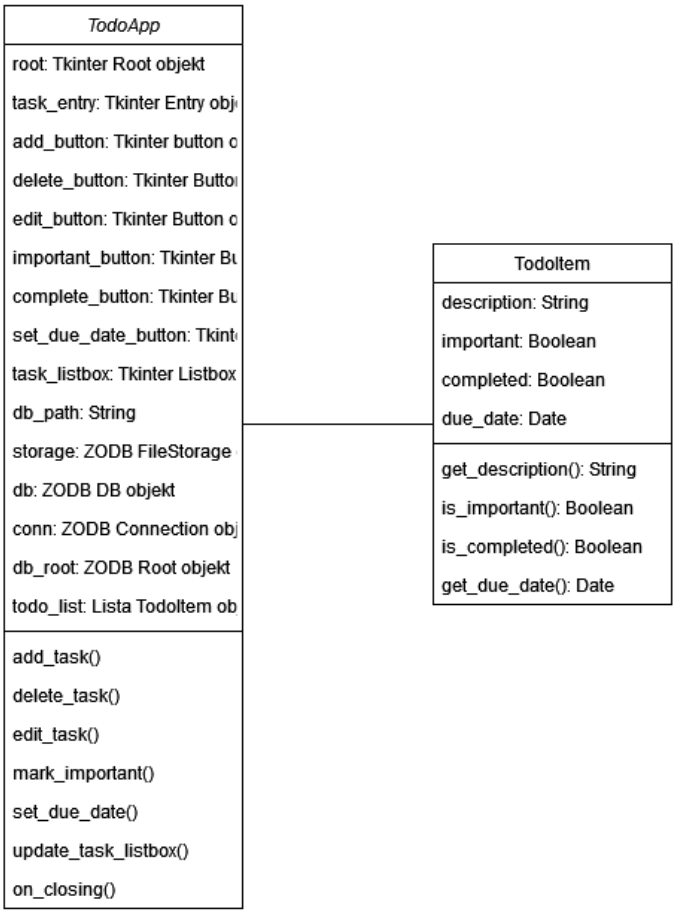
3.1. Arhitektura aplikacije

Arhitektura aplikacije za upravljanje zadacima temelji se na klasičnom modelu s grafičkim korisničkim sučeljem (GUI) i backendom koji komunicira s ZODB-om. Evo ključnih komponenti arhitekture:

- **Korisničko sučelje (GUI)**: Grafičko korisničko sučelje izrađeno je pomoću Tkinter biblioteke za Python. Ono omogućuje korisnicima unos, uređivanje i pregled zadatka.
- **Backend aplikacije**: Backend je odgovoran za komunikaciju s ZODB-om i logiku aplikacije. On sadrži klase za upravljanje zadacima i ostale potrebne komponente.
- **ZODB baza podataka**: ZODB se koristi za pohranu objekata zadatka i njihovih svojstava. Baza podataka čuva objekte u datotekama na disku pomoću FileStorage-a.

3.2. Dijagram klasa

Dijagram klasa vizualno prikazuje strukturu aplikacije i odnose između različitih klasa. Priloženi dijagram klasa jasno pokazuje kako su klase organizirane u aplikaciji i njihove metode.



Slika 1: UML dijagrama klasa (izvor: vlastita izrada)

3.3. Ključni dijelovi programskog koda

U ovom odjeljku, analizirat ćemo ključne dijelove programskog koda koji demonstriraju primjenu ZODB-a i objektno-orijentiranog pristupa. To uključuje:

- **Kreiranje i spremanje objekata zadatka u ZODB-u**: Kako se zadaci unose putem GUI-a, objekti se stvaraju i pohranjuju u ZODB bazu podataka.
- **Manipulacija objektima zadatka**: Prikazat ćemo kako se objekti zadatka uređuju, označavaju kao važni ili dovršeni te kako se postavljaju rokovi za zadatke.
- **Dohvat i prikaz objekata zadatka**: Objekti zadatka dohvaćaju se iz ZODB-a i prikazuju u korisničkom sučelju, omogućujući korisnicima pregled i interakciju s njima.

3.3.1. Kreiranje i spremanje objekata zadatka u ZODB-u

Prva i najviše korištena metoda u aplikaciji je kreiranje novih zadataka i spremanje istih u bazu. Metoda je definirana sljedećim kodom:

```
def add_task(self):
        description = self.task_entry.get()
        if description:
            due_date_str = simpledialog.askstring("Postavi rok", "Unesite
datum roka (dd.mm.yyyy):")
            if due_date_str:
                try:
                    due_date = datetime.strptime(due_date_str, "%d.%m.%Y")
                except ValueError:
                    messagebox.showerror("Greška", "Pogrešan format datuma.
Unesite prema formatu: dd.mm.yyyy.")
                    return
            else:
                due_date = None

            todo_item = TodoItem(description, due_date=due_date)
            self.todo_list.append(todo_item)
            self.update_task_listbox()
            self.task_entry.delete(0, tk.END)
            self.db_root['todos'] = self.todo_list
            transaction.commit()
```

Kôd 1.1. Metoda za kreiranje i spremanje objekata zadatka

Ovaj isječak koda odnosi se na funkciju add_task, koja se poziva kada korisnik želi dodati novi zadatak. Ključne točke su:

- Korisnik unosi opis zadatka putem grafičkog korisničkog sučelja.
- Zatim se korisniku traži unos datuma roka (ako želi postaviti rok).
- Nakon što se unese opis zadatka i, ako je potrebno, datum roka, stvara se nova instanca `TodoItem` klase (što predstavlja zadatak).
- Tako stvoren zadatak dodaje se u listu `self.todo_list`.
- Nakon dodavanja zadatka, poziva se funkcija `update_task_listbox` za ažuriranje prikaza zadatka u grafičkom korisničkom sučelju.
- Zatim se uneseni tekst iz polja za unos zadatka briše.
- Konačno, podaci se pohranjuju u ZODB bazu putem linije `self.db_root['todos'] = self.todo_list`, i transakcija se potvrđuje s `transaction.commit()`.

3.3.2. Manipulacija objektima zadatka

Za dodavanje oznake važnosti primjenjuje se sljedeća metoda:

```
def mark_important(self):
        selected_index = self.task_listbox.curselection()
        if selected_index:
            index = selected_index[0]
            if 0 <= index < len(self.todo_list):
                self.todo_list[index].important = True
                self.update_task_listbox()
                self.db_root['todos'] = self.todo_list
                transaction.commit()
```

Kôd 1.2. Metoda za dodavanje oznake „važno“

Ovaj isječak koda predstavlja funkciju `mark_important`, koja se poziva kada korisnik označi zadatak kao važan. Ključne točke uključuju:

- Funkcija provjerava koji zadatak je označen u grafičkom korisničkom sučelju, koristeći `self.task_listbox.curselection()`.
- Nakon što se odabere indeks označenog zadatka, provjerava se je li taj indeks valjan (0 <= index < len(self.todo_list)).

- Ako je indeks valjan, zadatak se označava kao važan tako što se postavlja svojstvo `important` na `True`.
- Zatim se poziva `update_task_listbox` za ažuriranje prikaza zadatka.
- Izmjene se pohranjuju u ZODB bazu i transakcija se potvrđuje.

3.3.3. Dohvat i prikaz objekata zadatka

Uzimanje svih zadataka iz baze i prikaz istih se provodi na sljedeći način:

```
def update_task_listbox(self):
        self.task_listbox.delete(0, tk.END)
        for item in self.todo_list:
            status = " (Bitno)" if item.important else ""
            status += " (Dovršeno)" if item.completed else ""
            due_date = item.due_date.strftime('%d.%m.%Y') if hasattr(item,
'due_date') and item.due_date else ""
            self.task_listbox.insert(tk.END, f"{item.description}{status}
{due_date}")
        transaction.commit()
```

Kôd 1.3. Dohvaćanje i prikaz zadataka u prozoru

Ovaj isječak koda predstavlja funkciju update_task_listbox, koja se koristi za ažuriranje prikaza svih zadataka u grafičkom korisničkom sučelju. Ključne točke uključuju:

- Postavke grafičkog prikaza (tkinter self.task_listbox) čiste se prvo (`self.task_listbox.delete(0, tk.END)`).
- Zatim se prolazi kroz sve zadatke u self.todo_list i za svaki zadatak izračunavaju se status (označava li se kao važan i/ili dovršen) i datum roka (ako je postavljen).
- Informacije o svakom zadatku dodaju se u grafički prikaz kao tekst.
- Na kraju, izmjene se pohranjuju u ZODB bazu i transakcija se potvrđuje.

3.4. Korištene biblioteke

Aplikacija koristi nekoliko ključnih biblioteka i ovisnosti koje trebaju biti instalirane prije pokretanja aplikacije. Popis korištenih biblioteka uključuje:

- tkinter: Biblioteka za izradu grafičkog korisničkog sučelja.
- ZODB: Biblioteka za rad s objektno-orijentiranom bazom podataka ZODB.

Kako biste instalirali ove biblioteke, možete koristiti alat za upravljanje paketima kao što je pip[7]. Primjer naredbi za instalaciju ovih biblioteka:

```
pip install tkinter
pip install ZODB
```

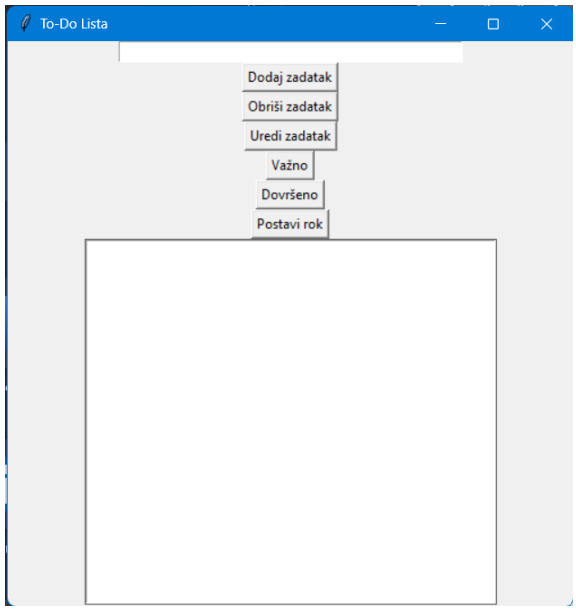
4. Primjer korištenja aplikacije

U ovom poglavlju, pružit ćemo detaljan prikaz kako koristiti našu aplikaciju za upravljanje zadacima koristeći objektno-orijentiranu bazu podataka ZODB. Opišemo osnovne korake i funkcionalnosti koje korisnici mogu očekivati prilikom korištenja aplikacije.

4.1. Početno sučelje

Kada prvi put pokrenete našu aplikaciju, prikazat će se početno sučelje koje omogućuje dodavanje, uređivanje i pregled zadataka. Ovdje su neki od ključnih elemenata sučelja:

- **Polje za unos zadatka**: Koristite ovo polje za unos opisa novog zadatka.
- **Gumb "Dodaj zadatak"**: Kliknite na ovaj gumb kako biste dodali uneseni zadatak na popis.
- **Popis zadataka**: Ovdje će se prikazivati svi vaši zadaci, zajedno s njihovim statusom (važno, dovršeno) i datumom roka (ako je postavljen).
- **Gumbi za akcije**: Postoje gumbi za označavanje zadatka kao važnog, dovršenog, uređivanje zadatka i brisanje zadatka.



Slika 2: Početni prozor aplikacije

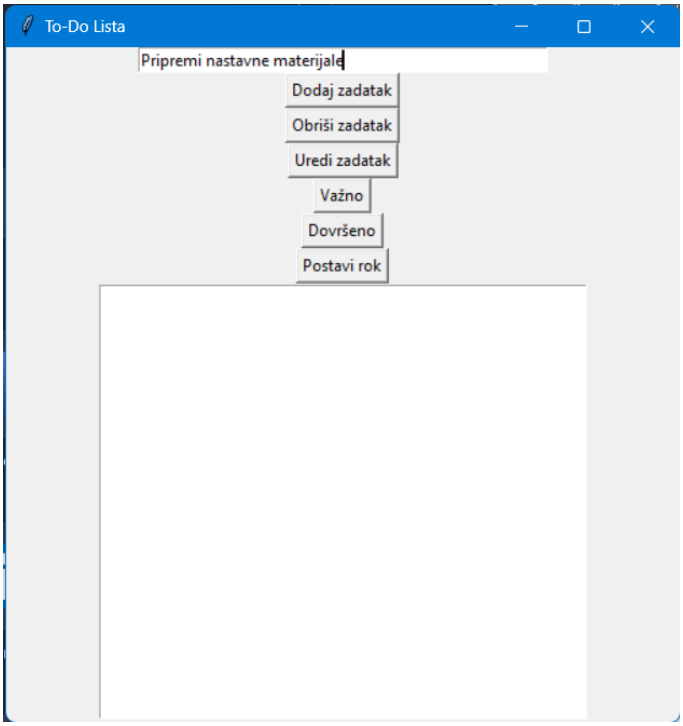
4.2. Dodavanje novog zadatka

Kod za kreiranje zadatka smo napisali u poglavlju 3.3.1.

Da biste dodali novi zadatak, slijedite ove korake:

1. Unesite opis zadatka u polje za unos.
2. Opcionalno, možete postaviti datum roka klikom na "Postavi rok" i unoseći datum u odgovarajućem formatu (dd.mm.yyyy).
3. Kliknite na "Dodaj zadatak" kako biste spremili novi zadatak.

Novi zadatak će se pojaviti u popisu zadataka.



Slika 3: Dodavanje zadatka

4.3. Uređivanje postojećeg zadatka

Metoda za uređivanje postojećeg zadatka glasi:

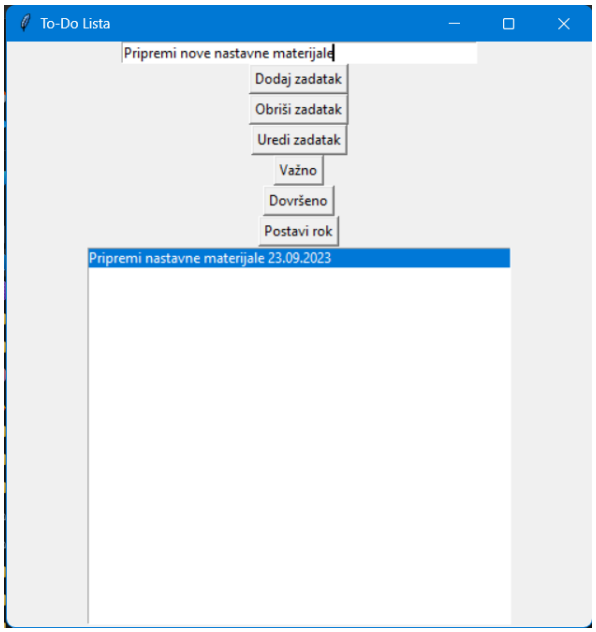
```
def edit_task(self):
        selected_index = self.task_listbox.curselection()
        if selected_index:
            index = selected_index[0]
            new_description = self.task_entry.get()
            if new_description:
                if 0 <= index < len(self.todo_list):
                    self.todo_list[index].description = new_description
                    self.update_task_listbox()
                    self.task_entry.delete(0, tk.END)
                    self.db_root['todos'] = self.todo_list
                    transaction.commit()
```

Kôd 1.4. Uređivanje postojećeg zadatka

Za uređivanje postojećeg zadatka, slijedite ove korake:

1. Odaberite zadatak iz popisa tako da ga kliknete.
2. Unesite novi opis zadatka u polje za unos.
3. Kliknite na "Uredi zadatak" kako biste spremili promjene.

Opis zadatka će se ažurirati u popisu.



Slika 4: Uređivanje zadatka

4.4. Označavanje zadatka kao važnog ili dovršenog

Metode za označavanje zadatka kao važnog je navedena u poglavlju 3.3.2., a za označavanje zadatka kao dovršenog je sljedeća:

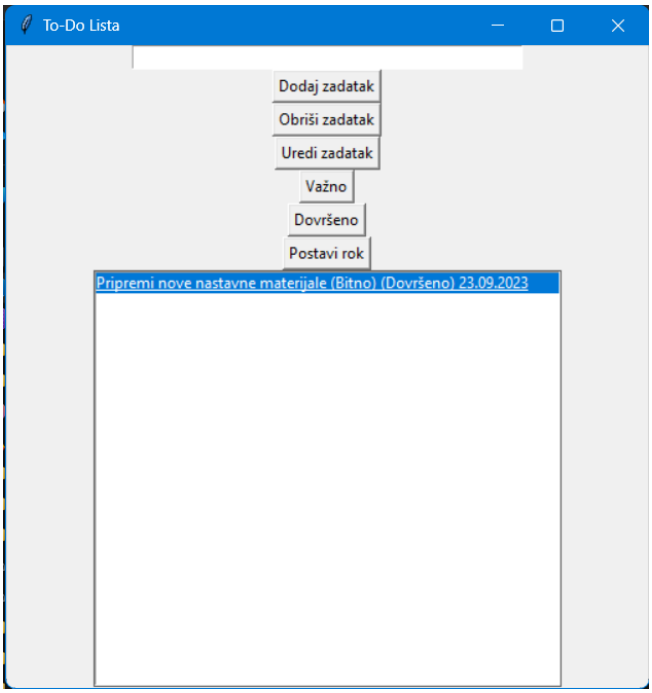
```
def mark_complete(self):
        selected_index = self.task_listbox.curselection()
        if selected_index:
            index = selected_index[0]
            if 0 <= index < len(self.todo_list):
                self.todo_list[index].completed = True
                self.update_task_listbox()
                self.db_root['todos'] = self.todo_list
                transaction.commit()
```

Kôd 1.5. Dodavanje oznake „dovršeno“

Da biste označili zadatak kao važan ili dovršen, slijedite ove korake:

1. Odaberite zadatak iz popisa tako da ga kliknete.
2. Kliknite na odgovarajući gumb "Važno" ili "Dovršeno" kako biste označili zadatak.

Status zadatka će se ažurirati u popisu.



Slika 5: Označavanje zadatka kao bitnog ili dovršenog

4.5. Postavljanje datuma roka

Kod za postavljanje datuma roka za izvršavanje zadatka glasi:

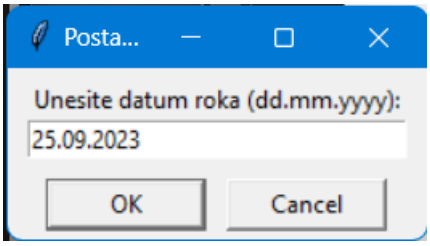
```
def set_due_date(self):
        selected_index = self.task_listbox.curselection()
        if selected_index:
            index = selected_index[0]
            task_item = self.todo_list[index]
            due_date_str = simpledialog.askstring("Postavi rok", "Unesite
datum roka (dd.mm.yyyy):")
            if due_date_str:
                try:
                    due_date = datetime.strptime(due_date_str, "%d.%m.%Y")
                except ValueError:
                    messagebox.showerror("Greška", "Pogrešan format datuma.
Unesite prema formatu: dd.mm.yyyy.")
                    return
            else:
                due_date = None

            task_item.due_date = due_date
            self.update_task_listbox()
            self.db_root['todos'] = self.todo_list
            transaction.commit()
```

Kôd 1.6. Postavljanje datuma roka

Za postavljanje datuma roka za zadatak, slijedite ove korake:

1. Odaberite zadatak iz popisa tako da ga kliknete.
2. Kliknite na "Postavi rok" i unesite datum roka u formatu dd.mm.yyyy.
3. Kliknite na "Postavi rok" kako biste spremili datum.

Datum roka će se prikazati uz zadatak u popisu.



Slika 6: Opcionalno postavljanje datuma roka

4.6. Brisanje zadatka

Metoda za brisanje zadatka s popisa i baze je sljedeća:

```
def delete_task(self):
        selected_index = self.task_listbox.curselection()
        if selected_index:
            index = selected_index[0]
            if 0 <= index < len(self.todo_list):
                del self.todo_list[index]
                self.update_task_listbox()
                self.db_root['todos'] = self.todo_list
                transaction.commit()
```

Kôd 1.7. Brisanje zadatka s popisa i iz baze

Za brisanje zadatka, slijedite ove korake:

1. Odaberite zadatak iz popisa tako da ga kliknete.
2. Kliknite na "Obriši zadatak" kako biste izbrisali zadatak.

Zadatak će biti uklonjen iz popisa.

Važno je napomenuti da se sve promjene automatski pohranjuju u objekno-orijentiranu bazu podataka ZODB.

5. Zaključak

Cilj ovog rada bio je razviti funkcionalnu aplikaciju i analizirati prednosti i ograničenja korištenja ZODB-a u kontekstu aplikacija za praćenje i organizaciju zadataka. Demonstrirano je kako se ZODB može koristiti kao pouzdana i učinkovita baza podataka za pohranu i upravljanje zadacima. ZODB je pružio jednostavan način za modeliranje i pohranu podataka u Pythonu. Identificirane su neke nedostatke i prostori za poboljšanja, uključujući bolju dokumentaciju, estetski privlačnije sučelje i napredne opcije pretrage i filtriranja. ovo istraživanje predstavlja početnu točku za razvoj aplikacija za upravljanje zadacima koristeći ZODB kao bazu podataka. Unatoč trenutnim ograničenjima, aplikacija pruža osnovne funkcionalnosti za praćenje i organizaciju zadataka. Daljnji razvoj i poboljšanja imaju potencijal za stvaranje snažnog alata za upravljanje zadacima.

6. Literatura

[1] Cattell, R. (1994). Object Data Management: Object-Oriented and Extended Relational Database Systems. Addison-Wesley.

[2] Atkinson, M.P., Bancilhon, F., DeWitt, D.J., et al. (1989). The Object-Oriented Database System Manifesto. In Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data.

[3] Maleković, M., Schatten, M., FOI (2017). Teorija i primjena baza podataka.

[4] ZODB Documentation. https://zodb.readthedocs.io/en/latest/ (pristupljeno 31.08.2023.)

[5] https://www.python.org/ (pristupljeno 31.08.2023.)

[6] https://docs.python-requests.org/en/latest/user/install/ (pristupljeno 31.08.2023.)

[7] https://www.tutorialspoint.com/python/python_gui_programming.htm/ (pristupljeno 31.08.2023.)

[8] https://pypi.org/project/pip/ (pristupljeno 31.08.2023.)

7. Popis slika

8. Popis isječaka programskog kôda

9. Prilog – programski kod

```
import tkinter as tk
from tkinter import simpledialog, messagebox
from ZODB import FileStorage, DB
import transaction
import persistent
from datetime import datetime
import os

class TodoItem(persistent.Persistent):
    def __init__(self, description, important=False, completed=False,
due_date=None):
        self.description = description
        self.important = important
        self.completed = completed
        self.due_date = due_date if due_date is not None else None

class TodoApp:
    def __init__(self, root):
        self.root = root
        self.root.title("To-Do Lista")
        self.root.geometry("500x500")
        self.db_path = os.path.expanduser("~/.skladiste_db.fs")
        self.storage = FileStorage.FileStorage(self.db_path)
        self.db = DB(self.storage)
        self.conn = self.db.open()
        self.db_root = self.conn.root()

        if 'todos' not in self.db_root:
            self.db_root['todos'] = []

        self.todo_list = self.db_root['todos']

        self.task_entry = tk.Entry(root, width=50)
        self.task_entry.pack()

        self.add_button = tk.Button(root, text="Dodaj zadatak",
command=self.add_task)
        self.add_button.pack()

        self.delete_button = tk.Button(root, text="Obriši zadatak",
command=self.delete_task)
        self.delete_button.pack()

        self.edit_button = tk.Button(root, text="Uredi zadatak",
command=self.edit_task)
        self.edit_button.pack()

        self.important_button = tk.Button(root, text="Važno",
command=self.mark_important)
        self.important_button.pack()

        self.complete_button = tk.Button(root, text="Dovršeno",
command=self.mark_complete)
        self.complete_button.pack()

        self.set_due_date_button = tk.Button(root, text="Postavi rok",
command=self.set_due_date)
```

```
        self.set_due_date_button.pack()

        self.task_listbox = tk.Listbox(root, selectmode=tk.SINGLE,
height=20, width=60)
        self.task_listbox.pack()

        self.update_task_listbox()

        self.root.protocol("WM_DELETE_WINDOW", self.on_closing)

    def add_task(self):
        description = self.task_entry.get()
        if description:
            due_date_str = simpledialog.askstring("Postavi rok", "Unesite
datum roka (dd.mm.yyyy):")
            if due_date_str:
                try:
                    due_date = datetime.strptime(due_date_str, "%d.%m.%Y")
                except ValueError:
                    messagebox.showerror("Greška", "Pogrešan format datuma.
Unesite prema formatu: dd.mm.yyyy.")
                    return
            else:
                due_date = None

            todo_item = TodoItem(description, due_date=due_date)
            self.todo_list.append(todo_item)
            self.update_task_listbox()
            self.task_entry.delete(0, tk.END)
            self.db_root['todos'] = self.todo_list
            transaction.commit()

    def delete_task(self):
        selected_index = self.task_listbox.curselection()
        if selected_index:
            index = selected_index[0]
            if 0 <= index < len(self.todo_list):
                del self.todo_list[index]
                self.update_task_listbox()
                self.db_root['todos'] = self.todo_list
                transaction.commit()


    def edit_task(self):
        selected_index = self.task_listbox.curselection()
        if selected_index:
            index = selected_index[0]
            new_description = self.task_entry.get()
            if new_description:
                if 0 <= index < len(self.todo_list):
                    self.todo_list[index].description = new_description
                    self.update_task_listbox()
                    self.task_entry.delete(0, tk.END)
                    self.db_root['todos'] = self.todo_list
                    transaction.commit()

    def mark_important(self):
        selected_index = self.task_listbox.curselection()
        if selected_index:
            index = selected_index[0]
            if 0 <= index < len(self.todo_list):
```

```
                self.todo_list[index].important = True
                self.update_task_listbox()
                self.db_root['todos'] = self.todo_list
                transaction.commit()

    def mark_complete(self):
        selected_index = self.task_listbox.curselection()
        if selected_index:
            index = selected_index[0]
            if 0 <= index < len(self.todo_list):
                self.todo_list[index].completed = True
                self.update_task_listbox()
                self.db_root['todos'] = self.todo_list
                transaction.commit()

    def set_due_date(self):
        selected_index = self.task_listbox.curselection()
        if selected_index:
            index = selected_index[0]
            task_item = self.todo_list[index]
            due_date_str = simpledialog.askstring("Postavi rok", "Unesite
datum roka (dd.mm.yyyy):")
            if due_date_str:
                try:
                    due_date = datetime.strptime(due_date_str, "%d.%m.%Y")
                except ValueError:
                    messagebox.showerror("Greška", "Pogrešan format datuma.
Unesite prema formatu: dd.mm.yyyy.")
                    return
            else:
                due_date = None

            task_item.due_date = due_date
            self.update_task_listbox()
            self.db_root['todos'] = self.todo_list
            transaction.commit()

    def update_task_listbox(self):
        self.task_listbox.delete(0, tk.END)
        for item in self.todo_list:
            status = " (Bitno)" if item.important else ""
            status += " (Dovršeno)" if item.completed else ""
            due_date = item.due_date.strftime('%d.%m.%Y') if hasattr(item,
'due_date') and item.due_date else ""
            self.task_listbox.insert(tk.END, f"{item.description}{status}
{due_date}")
        transaction.commit()

    def on_closing(self):
        self.conn.close()
        self.db.close()
        self.storage.close()
        self.root.destroy()

if __name__ == "__main__":
    root = tk.Tk()
    app = TodoApp(root)
    root.mainloop()
```