Alat za izradu online testova za Moodle sustav za e-učenje

Filip, Novački

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Filip Novački

ALAT ZA IZRADU ONLINE TESTOVA ZA MOODLE SUSTAV ZA E-UČENJE

DIPLOMSKI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Filip Novački

Matični broj: 44531/15–R

Studij: Informatika u obrazovanju

ALAT ZA IZRADU ONLINE TESTOVA ZA MOODLE SUSTAV ZA E-UČENJE

DIPLOMSKI RAD

Mentor :

Doc. dr. sc. Marcel Maretić

Varaždin, veljača 2024.

Filip Novački

Izjava o izvornosti

Izjavljujem da je moj diplomski 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

Sažetak

Rad opisuje biblioteku koja je napravljena u svrhu olakšavanja dijela nastavničkog procesa koji se bavi generiranjem pitanja koja se kasnije mogu izvesti u Moodle. Naglasak je stavljen na jednostavnost uporabe i proširivost.

Ključne riječi: moodle, ispitna pitanja, generiranje, python, cloze, kviz, automatizacija, nastava

Sadržaj

1. Uvod

U današnjem svijetu u kojem su računala postala dovoljno pristupačna i snažna da automatiziraju ogromnu količinu posla koje ljudi rade ponekad je nužno optimirati procese kako bi se pojednostavio ili ubrzao rad. U slučaju ovog rada, glavno područje optimizacije je nastava, odnosno stvaranje velikog broja ispitnih pitanja i pripadajućih odgovora.

Moodle je jedan od sustava za elektroničko učenje koji se koriste u nastavi ili kako bi se olakšala nastava uz korištenje IKT-a. Jedna od njegovih funkcionalnosti je i provjera znanja učenika koji ga koriste. Dakle, nastavnik unosi pitanja, odgovore te ostale parametre, a nakon što unese dovoljan broj pitanja, ona se daju učenicima koji ih onda rješavaju.

I dok je Moodle dobar kad je potrebno postojeću bazu pitanja promiješati, raspodijeliti i vrednovati, ima poprilično ograničene funkcionalnosti što se tiče generiranja velikog broja pitanja. Konkretno, u predmetima koji su ispunjeni računicom, Moodle ne zna generirati zadatke koji će imati pitanja i odgovore na neku klasu problema.

Biblioteka koja je predmet ovog rada riješit će upravo taj problem. Nastavnici će moći programatski generirati pitanja i njima pripadajuće odgovore. Rješenje je napraviti biblioteku koja je jednostavna za korištenje, koja se može dovoljno proširiti za one koji žele više funkcionalnosti te koje je dovoljno proširivo da se može povezati s drugim sustavima.

2. Opis problema generiranja testova

Sustav za e-učenje Moodle u sebi ima implementiran podsustav za testove. Taj sustav omogućava nastavnicima na predmetima da stvaraju testove, daju ih studentima na rješavanje te ih nakon toga sustav automatski ispravlja prema zadanim postavkama.

Sustav testova omogućava nastavnicima puno automatizacije, ali nije dovoljno robustan da podrži automatizaciju do razine da svaki student dobije jedinstven test, odnosno nije moguće parametarski odrediti pitanja i odgovore. Ono što olakšava pristup tom problemu je uvoz i izvoz pitanja iz vanjskih izvora, ali to zahtijeva postojanje baze pitanja u specifičnom formatu za pitanja.

Dodatni problem je što sintaksa za uvoz pitanja tih pitanja nije *čitka*. [1] Pitanje koje nosi 10 bodova, brojčanog je tipa, vrednuje odgovore $[4, 8]$ te ima posebne povratne informacije ovisno o odgovoru kojeg student upiše izgleda ovako:

```
Koliko je 5 + 2? {10:NUMERICAL:~4:#Pogresno~5:#Pogresno~6:#Blizu~=%100%7:#Tocno~8:#Previse}
```

Ova sintaksa nije toliko složena da bi bilo nemoguće dešifrirati o čemu se radi, ali je dodatni problem u tome što se u sintaksi može dogoditi greška, mogu se zabunom unijeti krivi podatci, a te će pogreške biti problematično detektirati. U konačnici, ako računalo može napraviti posao umjesto ljudi, zašto ne bi?

Bilo bi praktičnije kad bi prikaz ovog pitanja za nastavnika koji sastavlja testove izgledao otprilike ovako:

> PITANJE: KOLIKO JE 5 + 2?
>
> ODGOVORI:
>
> - 4 (POGREŠNO)
> - 5 (POGREŠNO)
> - 6 (BLIZU)
> - **7 (TOČNO)**
> - 8 (PREVIŠE)

Dodatna problematika kod generiranja testova iz područja matematike je što je ponekad potrebno pisati LaTeX kod, a bez provjere kako to izgleda je složeno i podložno pogreškama. Nastavnik uvijek može posegnuti za vanjskim alatima, ali to i dalje nije idealno rješenje jer se može dogoditi greška kod selidbe koda iz LaTeX sučelja u Cloze sintaksu, a i nije garantirano da će sve raditi kako treba.

Za rješenje tog problema može se iskoristiti sučelje Jupyter Notebooka koje podržava prikaz raznih standardnih formata, kao što su slike, HTML, stablasti prikaz JSON-a, LaTeX i mnogi drugi.

Cilj ovog rada je napraviti `python` biblioteku koja će generirati bazu pitanja koja se može uvesti u Moodle, a istovremeno i olakšati nastavnicima stvaranje pitanja kroz prikaz tih pitanja bez korištenja sintakse koja je nepregledna za čitanje.

3. Implementacija

S obzirom na specifičnost domene u kojem će se biblioteka koristiti, bilo je potrebno donijeti odluke kako pristupiti kodu da bude što primjenjiviji, korisniji, a da istovremeno netko tko će u budućnosti održavati taj kod može lako nadopuniti sve što će biti potrebno bez straha da neke druge funkcionalnosti prestanu raditi na željen način.

3.1. Korisnički zahtjevi

Kako bi biblioteka zadovoljila sve zahtjeve, morat će zadovoljiti uvjete:

- jednostavno korištenje
- mogućnost naprednog korištenja parametriziranjem metoda
- proširivost
- jednostavna distribucija

Jednostavno korištenje znači da za korištenje osnovnih funkcionalnosti ove biblioteke nije potrebno napredno znanje Pythona. To će omogućiti da svi koji sudjeluju u nastavnom procesu mogu na jednostavan način napraviti bazu pitanja za svoje studente.

Naravno, ta jednostavnost ne smije zatvoriti mogućnost da vještiji u Pythonu koriste napredne funkcionalnosti. Uz pametnu parametrizaciju se skup funkcionalnosti može bogato dopuniti i to može značajno utjecati na lakoću korištenja biblioteke.

Prve dvije točke podrazumijevaju da svaka metoda i funkcija iz biblioteke ima razumne i korisne `default` argumente. To znači da će bez parametriziranja funkcije raditi one funkcionalnosti koje bi se najčešće koristile. Svi `default` argumenti smiju se *pregaziti* te se tako ponašanje svih metoda i funkcija može proširiti po želji.

Taj pristup također omogućuje svima koji će kasnije održavati biblioteku da dodaju nove funkcionalnosti bez straha da će se neke osnovne funkcionalnosti pregaziti.

3.2. Paradigma pisanja koda

Projekt će biti pisan u kombinaciji objektno-orijentirane paradigme te funkcionalnog programiranja.

Kod korištenja dviju paradigmi dogodit će se sukob oprečnih dogmi pojedinih paradigmi te će se to riješiti na način da se uzme ono što je najkorisnije za kvalitetan kod koji neće postati *technical debt* nakon nekoliko dodatnih mogućnosti.

Ni u kojem trenutku ne smijemo zaboraviti da je objektno-orijentiranu paradigmu teško izbjeći jer je Python prvenstveno objektni jezik. Štoviše, objekti su u Pythonu svi osnovni tipovi kao što su `stringovi`, liste, `dictionaryji`, `tupleovi` itd.; klase, koje su tipa `type`; instance

klasi, koje su tipa te klase; funkcije, koje su tipa `function` itd. [2] Drugim riječima. u Pythonu je sve objekt.

3.2.1. Pristup objektno-orijentiranoj paradigmi

Važnost objektno-orijentiranog programiranja je to što pitanja koja će se sastavljati za testove su vrlo slična, odnosno dijele velik broj komponenti. Pitanja će imati zajedničke metode za prikaz i izvoz podataka u formate:

- JSON (`jsonpickle` biblioteka)
- konzolni ispis (`print`)
- prikaz JSON formata u stablastoj strukturi (korisno za prikaz u Jupyter Notebooku)
- pretvaranje objekata u JSON pickle format koji se sprema u varijablu u obliku stringa
- Cloze format koji je pogodan za uvoz u Moodle

Isto tako, ti objekti će imati i zajedničke metode za uvoz podataka. Oni će omogućiti da se identični objekti Python paketi mogu dobiti u nepovezanim interpreterima bez gubitka podataka. To će biti omogućeno kroz JSON pickle formate i kroz Cloze format. `JSON pickle` format će se unositi u obliku `stringa` ili `dictionaryja`, a Cloze format u obliku `stringa`.

Iz objektno-orijentirane paradigme će izbjegavati stanje koliko god je moguće. Stanja su problematična u trenutcima kad se nekoliko objekata poveže u strukturu i kad je teško identificirati koji su sve mogući izvori promjene nekog stanja. Iz tog razloga će se u stanje, odnosno u objektne varijable upisivati što je manje moguće podataka, a ako je promjena stanja neizbježna, onda će se to eksplicitno prikazati kroz potpise funkcija i metoda.

3.2.2. Pristup funkcionalnoj paradigmi

Python ima većinu svojstava funkcionalnog programiranja implementirano unutar sebe, [3] ali u mnogim slučajevima to nije optimalan način programiranja. Na primjer, u Pythonu je moguće napraviti rekurzije, ali se programer odgovara od korištenja rekurzija jer su one namjerno implementirane suboptimalno u Pythonu.

Python tome doskače širokim spektrom alata kao što su `map`, `reduce`, i drugi koji mogu zamijeniti rekurzije. Osim tih alata, Python se velikim dijelom oslanja na brze iteratore i generatore te i sam stvoritelj Pythona tvrdi da bi optimizacija rekurzija bila protivna filozofiji Pythona uz mnoge objektivne razloge koji bi više štetili nego koristili Pythonu. [4] [5]

Iz funkcijske paradigme preuzet će se pogled na funkcije kao na osnovne gradivne sustave te čistoća funkcija. Svojstvo čistosti funkcija se postiže ispunjavanjem dvaju kriterija:

- za iste ulazne argumente funkcija će vratiti iste izlazne argumente (potrebno je dodatno naglasiti da ne smije postojati stanje, lokalne varijable, statičke varijable ili bilo što drugo što bi moglo utjecati na ponašanje funkcija)

- funkcija nema nikakve nuspojave osim onoga što eksplicitno vraća

Ta dva uvjeta mogu se ispuniti kad se softver stvara u Pythonu, ali je često nepraktično. Na primjer, konektori na baze podataka često se postavljaju kao objektne varijable jer bi se u protivnom morale proslijeđivati kroz sve funkcije koje ih žele koristiti, a to bi bilo vrlo nezgrapno. Također, identifikatori se idiomatski postavljaju kao vrijednosti u varijable klase pa metode objekata njima pristupaju na taj način. Alternativa tome bi bila imati neki oblik prepoznavanja trenutne klase (npr. uzorak dizajna *factory*) i prema tome zaključivanje koji identifikator je odgovarajući.

Primjer korištenja identifikatora može se prikazati pomoću ovog primjera iz koda biblioteke:

```
@dataclass
class ClozeResponse(Cloze):
    points_worth: int
    possible_answers: list[ClozeAnswer] = field(default_factory=list)
    question_type: str = None

class Numerical(ClozeResponse):
    question_type = "NUMERICAL"

class Multichoice(ClozeResponse):
    question_type = "MULTICHOICE"
```

U tom primjeru klasa `ClozeResponse` predviđa postojanje neke `question_type` varijable, ali ju ne popunjava jer ne zna koja je. To prepušta svojim podklasama koje će to same popuniti, a osim toga će se ponašati kao i roditeljska klasa. Pristup tim varijablama radit će se pomoću naredbe `self.question_type`, a izbjeći će se punjenje potpisa funkcije argumentima koji se nikada ne mijenjaju.

U većini slučajeva kod će slijediti pravila:

- svaka varijabla koja se koristi će se proslijediti kao argument
- svaka promijenjena varijabla vratit će se kao rezultat funkcije

Drugim riječima, ne smije se dogoditi "čarobno" mijenjanje varijabli niti izvlačenje podataka koji mijenjaju rezultat funkcije. Taj pristup također olakšava i čitanje koda. Umjesto

```
self.add_numbers()
```

kod bi trebao izgledati ovako:

```
self.result = self.add_numbers(self.a, self.b)
```

Ovi primjeri van konteksta nisu jako smisleni, ali u složenijim sustavima takav pristup ima velike koristi. Donji isječak puno jasnije dočarava namjeru autora da zbroji brojeve `a` i `b` te ih zapiše u `result`. Gornji primjer može značiti bilo što i može imati posljedice koje nisu jasne sve dok programer ne ide detaljno gledati što svaka funkcija radi.

3.2.3. Dataclass klase

Kako bi se olakšalo bilježenje podataka u klase, koristit će se biblioteka `dataclasses`. [6] Ta biblioteka proširuje klase dekoriranjem tako što se automatski generiraju čarobne metode `__init__` (odnosno konstruktor), `__repr__` i druge.

Taj dekorator omogućava olakšano kreiranje klasa i pojednostavljuje kod tako što apstrahira metode koje su zapravo trivijalne. Trivijalni konstruktor zapisuje sve vrijednosti iz argumenata konstruktora u varijable koje pripadaju objektu koji je u nastajanju.

Osim toga, konstruktor zna provjeriti i tipove podataka, zna paziti na obavezne argumente te na taj način uvelike olakšava korištenje klasa ukoliko su one namijenjene samo za nošenje podataka.

3.2.4. Ostala obilježja koda

Sav kod u biblioteci je tipiziran. To znači da svaki objekt koji se stvara bi trebao imati pretpostavljen tip koji poprima, svi argumenti funkcije bi trebali imati definirani ulazni tip, a sve funkcije bi trebale imati definirani izlazni tip ukoliko postoji.

To olakšava razvoj u Pythonu jer stroga tipizacija nije nametnuta pravilima jezika, što znači da bi tzv. nestašni programer mogao lako pozvati metode koje ne pripadaju tipu (klasi) objekta kojeg poziva ili koristiti operatore između nekombatibilnih tipova ili bilo koju drugu greške koja se može dogodit ukoliko se korisnik ne pridržava tipova.

Za tipiziranje se inače koristi biblioteka `typing`, no u Pythonu verzije 3.11 je sintaksa za tipizaciju uvedena bez potrebe za uvođenjem vanjskog modula tako da se slobodno mogu tipovi bilježiti nad svim objektima. Primjerice:

```
def handle_exception(exception_message: str, raise_error: bool = False) -> int:
    # ...
```

To olakšava i IDE-ima prepoznavanje tipova pa se greške mogu detektirati i prije pokretanja nekog komada koda.

3.3. Sustav klasa

Klase ovog projekta organizirane su na način da se svaki sastavni dio Cloze pitanja ponaša na sličan način. Instance podklasi se međusobno ugnježđuju u strukture, najčešće liste.

Dakle, objekt koji sadrži tekst pitanja u sebi sadrži listu objekata koji nose odgovore. Svaki od nositelja odgovora u sebi sadrži listu odgovora.

3.3.1. Klasa `Cloze`

Osnovna klasa koja predstavlja majku klasu svih ostalih klasa je `Cloze`. Ta se klasa nikad ne bi trebala instancirati jer ne nosi nikakve domenske funkcionalnosti, već samo sadrži zajedničke funkcije koje se odnose na sve buduće podklase te neke funkcije koje koje su tehničke naravi, a pomažu kod ispravnog izvršavanja koda.

Drugim riječima, ova klasa se ponaša nalik na apstraktne klase. Jedini razlog zašto se nije posegnulo za funkcionalnostima pythona iz biblioteke `ABC` za apstraktne klase je to što instanciranje klase Cloze nema smisla te nema apstraktnih metoda koje bi trebalo osigurati da se implementiraju prije nego što se mogu koristiti.

Glavne funkcionalnosti za uvoz i izvoz pitanja sadržane su u ovoj klasi. To su metode `export` i `import_question`. Metoda `export` vraća prikaz klase u obliku JSON pickle. Metoda `import_question` inverzna je metoda metode `export`. Dakle, izlaz iz `export` metode može poslužit kao ulaz u metodu `import_question` te se dobije identična klasa od koje je krenuo izvoz.

3.3.2. Klasa `Cloze Question`

Klasa `Cloze Question` je podklasa klase `Cloze`, a u kontekstu Cloze pitanja predstavlja tekst pitanja. Ona ima dvije varijable: `question_text` te `responses`.

Varijabla `question_text` u sebi nosi pitanje koje će se prikazati studentima. Unutar tog stringa mogu se nalaziti i parametri koji se mogu kasnije popunjavati prema potrebama testa.

Varijabla `responses` sadrži listu `ClozeResponse` objekata, odnosno klase koja nosi odgovore. Kako ova varijabla može biti prazna, ali ipak treba biti tipa liste, kod mora zagarantirati da će se uvijek kreirati prazna lista prilikom instanciranja klase. Zbog načina na koji rade `default` argumenti u Pythonu, to se mora napraviti pomoću funkcije iz biblioteke `dataclasses field`. Argument toj funkciji je funkcija koja se ponaša kao `factory` za objekte. Za potrebu liste koristi se klasa `list`.

Kako bi parametrizacija pitanja bila moguća, potrebno je imati i metodu koja će pretvoriti *placeholder* oznake u stvarne vrijednosti. U tu svrhu napravljena je metoda `fill_question_parameters` koja zapravo samo sakriva metodu `format` klase `string`. Ova funkcija je napravljena tako da može primiti imenovane i neimenovane argumente te će dobro reagirati na oba unosa, kao što bi i metoda `format`.

Razlog iz kojeg je važno da postoji metoda `fill_question_parameters` jest ta što će svaki korisnik biblioteke znati što radi ta metoda kad joj vidi ime i argumente. Upravljanje varijablama objekta izvan klase koristeći metode vanjskih objekata (u ovom slučaju `string`) je vrlo podložno nehotičnim semantičkim greškama koje će teško biti odgonetnuti. Osim toga, kod izgleda urednije i jasnija je podjela poslova između klasa.

3.3.3. Klase `ClozeResponse`

Svaki zadatak u Moodlu može sadržavati više pitanja u sebi, odnosno na više mjesta student može unijeti svoje odgovore, ali da to i dalje spada pod isti zadatak. S obzirom na to da onaj tko rješava taj zadatak daje odgovore na ta pitanja, u jeziku Moodla to se zove Cloze Response, a klasa koja to modelira je `ClozeResponse`.

Odgovornost svakog odgovora je da zna koliko bodova nosi. Dakle, sastavljač testa će zadati broj bodova tipa `int`. Taj broj predstavlja najveći mogući broj bodova koji se može ostvariti. Odgovornost vrednovanja odgovora prepušta se klasi `ClozeAnswer`, odnosno to nije odgovornost pitanja.

Klasa `ClozeResponse` također ima i metodu `export_cloze_response`. Ona služi za izvoz podataka iz objekta u format koji je poznat Moodleu. Na primjer, pitanje koje nosi deset bodova, u kojem je 7 točan odgovor, te u kojem odgovori između 4 i 8 imaju povratnu informaciju ima izvor u ovom obliku:

```
{10:NUMERICAL:~4:0#Pogresno~5:0#Pogresno~6:0#Pogresno~=%100%7:0#Tocno~8:0#Pogresno}
```

Varijabla `question_type` govori klasi kako da donosi odluke o izgledu pitanja. To se uglavnom odnosi na format na koji izvozi te na pojedina ograničenja koja nose formati. Na primjer, klasa `Numerical`, koja nasljeđuje klasu `ClozeResponse`, nosi vlastitu implementaciju metode `add_possible_answer`. U njoj se provjerava jesu li svi predviđeni odgovori tipa nekog brojačnog tipa. Ukoliko nisu, neće se moći dodati taj odgovor. U budućnosti je predviđeno dodati još implementacija mogućih odgovora na sličan način kao što je dodana i klasa `Numerical`.

3.3.4. Klasa `ClozeAnswer`

Klasa `ClozeAnswer` u sebi nosi moguće, odnosno predviđene odgovore. Smisleno je da se među predviđenim odgovorima uvijek nađe barem jedan točan odgovor, odnosno Moodle treba prepoznati kako dati osobi koja rješava test maksimalni broj bodova ako ponudi točan odgovor.

Osim točnog odgovora, mogu se zadati i polutočni odgovori, a mogu se i dati netočni odgovori. Svi odgovori koji se ne zadaju kao predviđeni Moodle će nagraditi s 0 bodova. Drugim riječima, korisnik ne mora zadati sve moguće odgovore na prepoznavanje. U praksi bi se ovdje trebali dodati točni odgovori, odgovori koji su polutočni (ako postoje), te česti netočni odgovori. Uz česte netočne odgovore može stajati i objašnjenje što je bilo pogrešno kako bi osoba koja je rješavala test mogla znati u čemu je pogriješila.

Sadržaj odgovora, odnosno vrijednost koja se uspoređuje s unosom osobe koja rješava test, postavlja se u varijablu `answer_text`. Vrijedi naglasiti da ta varijabla očekuje objekt tipa `str`, dakle čak i ako je odgovor numerički, očekuje se string. Provjera točnosti unosa kod numeričkih odgovora radi se tako što se ta varijabla pokuša pretvoriti u `float` i ako ne prođe, znači da je unos ne validan.

Točnost odgovora određuje se ponderom, odnosno varijablom `ponder`. Ukoliko se ponder ne zada, klasa će pretpostaviti da je 1. Ponder se ponaša kao faktor. Ako je ponder 1, onda

se dobivaju svi bodovi. Ukoliko je 0, ne dobivaju se bodovi. Zbog načina na koji se ponaša Moodleov format za pitanja, taj se faktor pretvara u postotak. Dakle, ponder koji je zadan kao 1 u izvozu će izgledati kao 100. Pomoću pondera je predviđeno određivati točne, polutočne i netočne odgovore.

Varijabla `feedback` je povratna informacija koja će se prikazati osobi koja rješava test ako ponudi odgovor koji se poklapa s vrijednosti zadanom u `answer_text`. Ispunjavanje te vrijednosti nije obavezno, ali će sastavljač testa najbolje znati kad je potrebno nešto sugerirati osobi koja rješava test.

Varijabla `tolerance` korisna je ukoliko zadatak nosi rizik da se dogodi mala greška, npr. pri zaokruživanju ovisno o načinu rada. Tip koji se očekuje u ovom polju je `float`, a ponaša se tako da se ponuđen odgovor usporedi s varijablom `answer_text` te se provjeri je li odgovor udaljen za manje od vrijednosti `tolerance` od `answer_text`. Ukoliko se ne zada tolerancija, Moodle će pretpostaviti da je 0, odnosno da odgovor mora biti identičan kao i ponuđen odgovor.

U klasi `ClozeAnswer` postoji i varijabla `is_correct` koja označava je li odgovor točan, a ponaša se slično kao i kad se ponder postavi na 1. To omogućuje da sastavljač postavi neko ponder na 1 na neki odgovor, ali da i dalje to ne bude točan odgovor.

Važno je napomenuti da implementacija softvera ne bi trebala ograničiti korisnike od onoga što u nekim slučajevima možda ima smisla, ali također se korisnici pozivaju da donose mudre odluke u dizajnu svojih pitanja.

4. Korištenje biblioteke

Kod dizajna ove biblioteke vodilo se računa o tome da je sučelje prema korisniku bude što jednostavnije i očitije. Dakle, nije potrebno napredno znanje Pythona, a poruke upozorenja i pogrešaka bi trebale biti očite kako bi korisnik bez problema znao gdje je problem.

4.1. Uvod u sastavljanje pitanja

Sastavljanje pitanja počinje ovako:

```
from cloze.cloze_question import ClozeQuestion

question = ClozeQuestion(
    "Trokut ima katete duljine 3 i 4. Kolika je njegova dijagonala?"
)
```

Time je sastavljeno pitanje. To pitanje još nije u funkcionalnom stanju jer nema moguće odgovore, bodove i ostale parametre koji su potrebni kako bi bilo potpuno.

Dodajmo polje za odgovor pitanju.

```
from cloze.cloze_question import ClozeAnswer, Numerical

response = Numerical(1)  # pitanje je numerickog tipa i nosi 1 bod

response.add_possible_answer(
    ClozeAnswer(answer_text="5", is_correct=True)
)

question.add_response(response)
```

Time je dodan mogući odgovor "5" koji je točan. Kako bi se to pitanje moglo uvesti u Moodle, potrebno je napraviti izvoz, odnosno export.

```
question.export_cloze_question()
```

Rezultat će izgledati ovako:

```
{1:NUMERICAL:~=%100%5:#~}
```

4.2. Uvoz i izvoz podataka

Kako bi se koristila funkcionalnost pomoću koje više korisnika može uređivati isti skup pitanja, korisnik treba uvoziti i izvoziti pitanja. Za demonstraciju ove funkcionalnosti bit će iskorišteno pitanje iz prethodnog poglavlja.

```
question.export()
```

Ukoliko se koristi metoda `export` bez argumenata, ona će se ponašati kao da ima proslijeđen enumerator `ExportType.IPY_DISPLAY`. To govori programu da u interaktivnu ljusku (*shell*) ispiše JSON koji se može lako pregledavati.

Naravno, ukoliko se ne koristi interaktivna ljuska, ta vrsta izvoza neće uspjeti i dignut će se iznimka.

U slučaju da korisnik želi napraviti sličnu vrstu izvoza, ali program se ne koristi u interaktivnom sučelju, može se koristiti enumerator `ExportType.PRINT`. Tada će izvoz biti čitljiv korisniku i "lijepo" oblikovan, ali će istovremeno ispisati poruku direktno u konzolu u kojoj se vrti biblioteka.

Uz manje modifikacije, moguće je napraviti i bilježenje (*logging*) kako bi se u većim sustavima moglo pratiti ponašanje biblioteke.

Treća vrsta izvoza je u obliku JSON-a koji se postiže uz argument `ExportType.JSON`. U tom slučaju će poziv metode vratiti JSON string koji se kasnije može uvesti. Opisan tok podataka funkcionira na sljedeći način:

```
question_export = question.export()

imported_question = Cloze.import_question(question_export)
```

Nakon ovog postupka, u varijablama `question` i `imported_question` bit će zapisana ista pitanja. S obzirom da čarobna metoda `__eq__` nije implementirana, testiranje jednakosti neće biti moguće sve dok se ta metoda ne implementira.

JSON oblik zapisa omogućuje i olakšanu kooperaciju tako da taj string može iskoristiti i netko drugi, uz jedini uvjet da je ova biblioteka instalirana.

4.3. Korištenje parametriziranih pitanja i odgovora

Kako bi se neko općenito pitanje moglo parametrizirati, potrebno je zadati pitanje koje podržava parametrizaciju. Metoda `fill_question_parameters` klase `ClozeQuestion` je zapravo samo maska oko string metode `format`. Drugim riječima, svi argumenti koji se proslijede metodi `fill_question_parameters` će se proslijediti u metodu `format`.

To također znači da i pitanje treba biti oblikovano način na koji se stringovi inače pripremaju na formatiranje. Primjerice:

```
question = ClozeQuestion("Koliko je {} + {}?")
```

Unutra možemo staviti bilo koje varijable. U ovom primjeru bi imalo smisla popuniti ih brojevima.

```
question.fill_question_parameters(5, 2)
```

Tako pitanje se pitanje iz koje je spremno na parametrizaciju, dakle `Koliko je {} + {}?` pretvara u `Koliko je 5 + 2?`

4.4. Dodavanje prostora za odgovor i usklađivanje s parametriziranim pitanjima

Pitanje iz prethodnog pitanja treba imati prostor za odgovor. Kako bi se to ostvarilo, potrebno je prvo instancirati bilo koju podklasu od ClozeResponse. U ovom slučaju to će biti Numerical. Nakon toga je potrebno dodati odgovore i sve ih pripasati istom pitanju. Dakle,

```
response = Numerical(10, question_type="NUMERICAL")
for answer in range(10):
    ponder = 0
    feedback = "Krivo"
    if answer == 7:
        ponder = 1
        feedback = "Tocno"

    response.add_possible_answer(
        answer_text=str(answer),
        ponder=ponder,
        feedback=feedback,
        is_correct=bool(ponder)
    )
question.add_response(response)
```

Ovaj pristup je koristan ukoliko se treba napraviti jedno pitanje. Kako bi se generiralo više pitanja i odgovora, moguće je napraviti niz pitanja kroz sustav petlji koje će činiti generator. Primjerice:

```
def return_sum_questions(number_of_questions: int, question_object: type[Cloze]):
    for _ in range(number_of_questions):
        for a in range(10):
            for b in range(10):
                response = Numerical(10, question_type="NUMERICAL")
                response.add_possible_answer(
                    ponder = 0,
                    feedback = "Krivo",
                    answer_text = str(a - b),
                    is_correct = False,
                )
                response.add_possible_answer(
                    ponder = 1,
                    feedback = "Ispravno",
                    answer_text = str(a + b),
                    is_correct = True,
                )
                new_question = question_object.fill_question_parameters(a, b)
                new_question.add_response(response)
                yield new_question
```

Ovaj primjer više je demonstrativnog karaktera. Unutar tih petlji mogu se dodati razne provjere kako rezultati ne bi bili nemogući, preteški ili neprikladni za određene klase problema. Npr. ukoliko se treba dogoditi neko dijeljenje, može se napraviti provjera da se u nazivniku nikad ne nalazi nula, što je moguće s nekim kombinacijama brojeva.

Ovo su osnove za generiranje velikog broja zadataka. Više parametara će vjerojatno značiti više `for` petlji, dok će složenije klase zadataka značiti složeniji uvjeti koje brojevi moraju zadovoljiti kako bi zadatak biti validan.

5. Moguće nadogradnje

Sustav koji je napravljen u sklopu ovog rada sadrži osnovne funkcionalnosti za rad. Također, pri stvaranju biblioteke vođeno je računa o tome da se lako mogu dodati nove funkcionalnosti, odnosno da je kod dovoljno modularan da se mogu dodati nove funkcionalnosti prema potrebi.

5.1. Sustav upozorenja

Sustav upozorenja bi bila nadogradnja koja bi olakšala korisniku da rad na način da eliminira "glupe" greške, odnosno greške koje slučajno mogu nastati kao greška u pisanju i nemaju smisla.

5.1.1. Dva identična odgovora

Primjerice, nikad isto pitanje ne bi trebalo imati dva ista odgovora. Moguće je da sastavljač pitanja napravi grešku nesvjesno ili krivom uporabom biblioteke. Sustav bi u tom slučaju detektirao postojanje već istog pitanja te bi javio korisniku da to vjerojatno nije mudro.

To se može implementirati na više načina. Jedan način je da se kod unosa odgovora provjeri postoji li već isti odgovor na to pitanje. Ukoliko postoji, može se ispisati ili logirati upozorenje, a može se i dignuti iznimka. Ispis upozorenja je puno lakše naravi, odnosno korisnik može to vrlo lako ignorirati, dok se dizanje iznimke može isključivo eksplicitno ignorirati na način da se ta greška ulovi i da se nastavi rad programa.

5.1.2. Minimalno jedan točan odgovor

Također, svako bi pitanje trebalo imati označen jedan odgovor kao točan. Korisnik kod sastavljanja pitanja može slučajno zaboraviti označiti odgovor kao točan te na taj način zadati glavobolje onome tko će rješavati test.

S obzirom na to da bi ovaj nedostatak točnog odgovora mogao donijeti problema kasnije i kod uvoza u Moodle, najbolje bi bilo da se korisnika na ovo upozori dizanjem iznimke kod izvoza. Tako se može izbjeći da sastavljač pitanja napravi grešku gdje ima pitanja bez točnog odgovora.

5.1.3. Prazna pitanja i odgovori

Slučajno bi se sastavljačima pitanja moglo dogoditi da zaborave unijeti pitanje ili da unesu odgovor koji je prazan string. Ta provjera bi bila vrlo jednostavna jer korisnik može dobiti upozorenje čim unese pitanje ili odgovor jer se unosi mogu odmah validirati.

Kako je ovo problem koji se može naknadno riješiti, ispis upozorenja bi potencijalno bilo dovoljno dobro rješenje, ali također se može napraviti i dizanje iznimke kako bi se zaustavilo

izvršavanje programa, pogotovo ako je puno pitanja generirano odjednom.

5.2. Veći broj vrsta pitanja

Moodle trenutno podržava osam različitih vrsta pitanja:

- `SHORTANSWER` ili kratko `SA`, odnosno kratki odgovori
- `SHORTANSWER_C` ili kratko `SAC`, odnosno kratki odgovori osjetljivi na kapitalizaciju slova
- `NUMERICAL` ili kratko `NM`, pitanja s brojčanim odgovorima
- `MULTICHOICE` ili kratko `MC`, pitanja s više mogućih odgovora, od kojih je jedan točan, u tri inačice:
  - s padajućim izbornikom
  - sa stupcem *radio* gumbića
  - s retkom *radio* gumbića
- `MULTIRESPONSE` ili kratko `MR`, pitanja s više mogućih odgovora i s više mogućih točnih odgovora, u dvije inačice:
  - s retkom gumbića
  - sa stupcem gumbića

Za potrebe ovog rada napravljena je implementacija samo za `NUMERICAL` pitanja, a ostala vrsta pitanja se mogu implementirati. Za svaki od dodatnih vrsta pitanja neće biti potrebne velike intervencije jer su osnove klasa već napravljene, samo je potrebno napraviti klase za pojedinu vrstu pitanja, pregaziti metode ukoliko imaju posebne funkcionalnosti te dodati specifičnosti koje su posebne za svaku vrstu pitanja.

Kod je predviđen s tim dodatkom u vidu tako da strukturalne promjene neće biti potrebne, ili ako da, minimalne.

5.3. Automatsko generiranje odgovora

Kako se klase problema koja se rješavaju u predmetima uglavnom ne mijenjaju previše, korisno bi bilo implementirati automatsko generiranje pitanja i odgovora ovisno o tome koja se klasa zadatka traži.

To bi bio popriličan zahtjev zbog velikog broja klasa zadataka koje se trebaju riješiti, ali ponovno iskorištavanje starog koda za generiranje zadataka je svakako prihvatljiva opcija.

To znači da umjesto da korisnik mora pisati cijeli program sam koji će generirati pitanja i odgovore, umjesto toga može samo odabrati klasu problema i pitanja će biti generirana.

5.4. Više tipova izvoza i uvoza

Za potrebe ovog rada napravljen je minimum tipova izvoza kako bi biblioteka bila funkcionalna. To su:

- Moodle Cloze format za pitanja
- `jsonpickle`
- ispis u aktivno sučelje

Moodle Cloze format je izvoz točno u onom formatu koji se može uvesti u Moodle, točno prema definiciji kakvu Moodle daje. On je nužan kako bi se ostvarilo sučelje između biblioteke i Moodlea.

Izvoz u `jsonpickle` je koristan jer omogućuje da se pitanje spremi u tekstualni format koji se može spremiti u čitljiv tekstualni dokument, a može se na jednostavan način ponovno učitati u sustav klasa. To olakšava rad s pitanjima ukoliko je predviđeno da više korisnika radi na istim pitanjima.

Ispis u aktivno sučelje je zapravo vizualna podrška sastavljaču pitanja. Ukoliko se biblioteka pokreće u Jupyter okruženju, onda se koristi LaTeX za ispis te ispravno formatiranje, a ako se biblioteka pokreće u običnom *shell* okruženju, onda se radi rudimentarni ispis kako bi korisnik vidio opće informacije o pitanju.

5.4.1. Datoteke iz tabličnih aplikacija

Ovdje se otvara prilika da se izvoz napravi u neki od formata koji je koristan, a trenutno nije napravljen. Ovisno o potrebama korisnika, to može biti izvoz/uvoz iz/u Excel tablice/u. Tako će korisnik moći sastaviti pitanja u Excelu (ili ih ima sastavljene od prije), pa se mogu parsirati te tako uvesti u biblioteku.

Nedostatak ovog pristupa je što povijesno gledano datoteke iz Microsoftovog Office paketa nisu bile konzistentnog formata (čak nisu niti unatrag kompatibilne), što bi značilo da će biblioteka tako imati snažnu ovisnost prema drugoj biblioteci u moći će raditi isključivo sa zadnjim verzijama tih pomoćnih biblioteka.

5.4.2. Manipulacija CSV datotekama

Alternativa datotekama iz tabličnih aplikacija je korištenje CSV datoteke za uvoz i izvoz. Svaka aplikacija koja radi na principu tablica može uvoziti i izvoziti u CSV format, a kako je to vrlo jednostavan *plain text* format, može se i lako koristiti za uvoz i izvoz.

Parsiranje CSV datoteka može se napraviti ručno na jednostavan način, a može se i koristiti biblioteka u Pythonu koja to već radi automatski.

Kako je CSV tako jednostavan format, neće se dogoditi da netko promijeni izgled tog formata te da se stvori snažna ovisnost prema nekoj drugoj biblioteci.

5.5. Komunikacija s bazom podataka

Pitanja se također mogu crpiti i spremati u bazu podataka. ERA model se može napraviti slično prema modelu klasa iza aplikacije, ili se može iskoristiti neka nerelacijska, odnosno dokument baza.

Snažne veze između relacija nisu potrebne jer vjerojatno neće biti potrebno pretraživati često po pitanjima, broju bodova i ostalim parametrima. Na zahtjev prema pitanju se mogu dohvatiti sve informacije o tom pitanju, uključujući i odgovore, u nekom od formata koji je standardiziran i već iskorišten, npr. `jsonpickle`.

Komunikacija s bazom omogućuje i daljnji razvoj oko ove biblioteke, npr. u smjeru web aplikacije.

5.6. Web aplikacija

Kako bi se koristila biblioteka u njenom sadašnjem stanju, potrebno je donekle poznavati Python, razvojna okruženja, i donekle biti pismen oko programiranja. To se u nekim slučajevima može očekivati, ali u nekim područjima je to previše za očekivati on onih koji sastavljaju pitanja.

5.6.1. Frontend

Korisnicima koji nisu vični bijelom tekstu na crnoj pozadini bilo bi daleko jednostavnije kad bi mogli otvoriti web preglednik i onda unositi pitanja i odgovore u "lijepo" sučelje.

To se može jednostavno realizirati uz frontend koji će podržavati unos pitanja, odgovora, bodova i slično. Kako je ova biblioteka napravljena relativno jednostavno, odnosno bez pakla velikih ovisnosti prema drugim bibliotekama, bilo bi idealno kad bi se ta jednostavnost mogla održati i u ostalim dijelovima te aplikacija. Zato se savjetuje potencijalnom autoru frontenda da ne poseže za megalomanskim razvojnim okvirima za JavaScript, nego da se drži nečeg jednostavnog kako bi aplikacija bila jednostavna, a samim time vjerojatno i responzivnija i lakša za koristiti.

Uz to se preporučuje i Elm, funkcijski frontend jezik koji se kompajlira u Javascript, a napravljen je tako da eliminira sve *runtime* greške koje se mogu dogoditi, a ujedno je izuzetno brz i jednostavan za programiranje. [7]

5.6.2. Backend

Frontend bi teško mogao funkcionirati bez kvalitetnog backenda jer pri svakom ponovnom pokretanju stranice bi se trebala sva pitanja ponovno ispunjavati. Tome se može doskočiti pomoću kolačića i sličnim trikovima, ali daleko je od pravog backenda koji će dohvaćati sve podatke.

Backend bi mogao iskoristiti funkcionalnost uvoza i izvoza raznih formata iz biblioteke

te ih spremati u vanjsku bazu podataka, a prema potrebi ih dostavljati prema frontendu. To bi korisnicima omogućilo da vrlo lako mogu dolaziti do svojih pitanja koja su kreirali, čak i na drugim računalima, a eliminira se mogućnost da zbog korisničke pogreške (npr. slučajno brisanje podataka) nestanu pitanja koja je možda korisnik godinama sastavljao.

Nešto snažniji backend bi mogao omogućiti i većem broju korisnika da rade nad istim skupom pitanja, što može uvelike olakšati rad u sustavima gdje je više nastavnika na istom predmetu. Također, snažan backend bi se mogao povezati direktno s Moodleom i omogućiti da korisnik ne mora nikakav ručni posao raditi, nego da jednostavno iz aplikacije uveze pitanja u Moodle.

Ovisno o potrebi projekta, backend se može napraviti vrlo različitih složenosti, a također ima i veliku širinu funkcionalnosti koje može pokriti.

6. Zaključak

Kod pripremanja kvizova za učenike, skalabilnost je veliki izazov, pogotovo kad je broj učenika velik. Ista pitanja se brzo počinju ponavljati, učenici skupljaju fond pitanja i ponekad je izazovno imati svježa pitanja za učenike.

Tome u spas dolazi biblioteka koja je glavna tema ovog rada. Ona omogućuje generiranje beskonačno mnogo pitanja i pripadajućih odgovora, odnosno barem u gabaritima koje može podržati snaga računala i Moodle.

Kako bi tu biblioteku moglo koristiti što više ljudi, odnosno što veći broj nastavnika, posebno je vođeno računa o tome da je sučelje za korištenje čim jednostavnije, odnosno da postoje razumne pretpostavljene postavke za jednostavno korištenje. Istovremeno, korisnici mogu značajno proširiti mogućnosti tako što se parametriziraju razne opcije te se tako mogu postići i razne opcije koje podržava Moodle.

Naravno, ova biblioteka nije zamišljena da je početak i kraj cijelog ekosustava pitanja. Dodatno olakšanje korištenja može se napraviti ako se biblioteka poziva kao dio nekog šireg sustava, npr. s frontendom ili s bazom. Tako se može omogućiti da ju koriste i korisnici koji uopće ne znaju programirati.

Kod razvoja se vodilo računa i o tome da se sustav može proširiti. Dakle, kod je jednostavan, eksplicitan i naglasak je bio na tome da se što lakše može prepoznati o čemu se radi u kojem dijelu koda.

Autor ove biblioteke, unatoč moći koju ona pruža, želi da njeni korisnici savjesno koriste njene funkcionalnosti i da ne zaborave da su kvaliteta pitanja i odgovora puno važniji nego njihova kvantiteta. Korištenje masovnog generiranja pitanja može se pretvoriti u tiraniju, stoga vodimo računa da naše učenike prije svega motiviramo na učenje i na kvalitetan rad, a da tek onda posegnemo za alatima koji uistinu pomiču granice u kontekstu ispita i provjera znanja.

Popis literature

[1] Moodle, „Embedded Answers (Cloze) question type," adresa: https://docs.moodle.org/402/en/Embedded_Answers_(Cloze)_question_type (pogledano 29. 6. 2024.).

[2] P. S. Foundation, „Built-in Types," adresa: https://docs.python.org/3.11/library/stdtypes.html (pogledano 29. 6. 2024.).

[3] P. S. Foundation, „Functional Programming HOWTO," adresa: https://docs.python.org/3.11/howto/functional.html (pogledano 29. 6. 2024.).

[4] G. van Rossum, „Tail Recursion Elimination," adresa: https://neopythonic.blogspot.com/2009/04/tail-recursion-elimination.html (pogledano 29. 6. 2024.).

[5] G. van Rossum, „Final Words on Tail Calls," adresa: http://neopythonic.blogspot.com/2009/04/final-words-on-tail-calls.html (pogledano 29. 6. 2024.).

[6] P. S. Foundation, „dataclasses – Data Clases," adresa: https://docs.python.org/3.11/library/dataclasses.html (pogledano 29. 6. 2024.).

[7] E. Czaplicki, „elm," adresa: elm-lang.org (pogledano 29. 6. 2024.).

1. Prilog 1: poveznica na repozitorij

Repozitorij s kodom od biblioteke o kojoj se radi dostupan je na: `https://github.com/filipnovacki/jocular`