UNIVERZITET U NOVOM SADU
PRIRODNO-MATEMATIČKI
FAKULTET
DEPARTMAN ZA MATEMATIKU
I INFORMATIKU

Smerač

- seminarski rad iz predmeta Skript jezici -

Aleksa Siriški, 159/22
Novi Sad, 2022.

Sadržaj

1 Uvod

1.1 Discord

Discord[1], kao jedna od najpopularnijih društvenih mreža za programere i entuzijaste računarskih tehnologija, je logičan izbor za razmenu informacija i raspoređivanje časova IT i RN smerova Univerziteta u Novom Sadu. Uprkos jednostavnosti Discord-ovog grafičkog interfejsa nije lako omogućiti studentima da sami sebi određuju smer a direktna integracija sa Google kalendarom je apsolutno neizvodljiva. Na sreću, Discord tim je osposobio trećim licima da kreiraju 'Bot' naloge i uz pomoć njihovog REST API će nastati Smerač - Discord Bot stvoren da omogući studentima izbor sopstvenog smera kao i integraciju Google kalendara.

1.2 Python

Smerač je Python[2] skripta napisana u svega 300 linija koda. Računajući da korisnicima nije bitno da li će im smer biti dodeljen za 1ms ili 1s Python je bio prikladan izbor. Najpre zbog mnogobrojnih ugrađenih biblioteka i modula, kao što su interactions-py[3] i asyncio[4], ali i zbog jednostavne sintakse koja je omogućila eksponencijalan razvoj ovog programa.

1.3 Ideja

Discord bot koji mora biti dovoljno jednostavan ali isto tako i univerzalno programiran da se može lako primeniti na više različitih okruženja (Discord servera). Takva prilagodljivost se postiže pametnim planiranjem toka rada programa i korišćenjem promenljivih okruženja (eng. *environment variables*). Takođe je jako bitna licenca koja će se koristiti, da ne bi došlo do krađe autorskih prava i zloupotrebe softvera. Tačno iz tih razloga odabrana licenca za Smerač će biti GNU General Public License[5].

1.4 Asinhronizacija

Kako bi program mogao da istovremeno čita poruke od različitih korisnika potrebno je kreiranje dodatnih niti. Pomoću biblioteke asyncio Smerač će imati mogućnost paralelnog parsiranja kalendara i dodeljivanja smera svakom studentu koji to zatraži. Pored bržeg izvođenja programa asinhrone funkcije služe da oduzmu čekanje odgovora od servera, tj. ako zahtev prvog korisnika traje duže da se obradi drugi korisnici ne moraju da čekaju nego se i njihovi zahtevi obrađuju u istom trenutku.

1.5 Smerovi

Najbitnija i najjednostavnija funkcionalnost Smerača će biti dodeljivanje smera studentima. Program će izvršavati korisničke komande oblika:

`/smer <naziv_smera>`

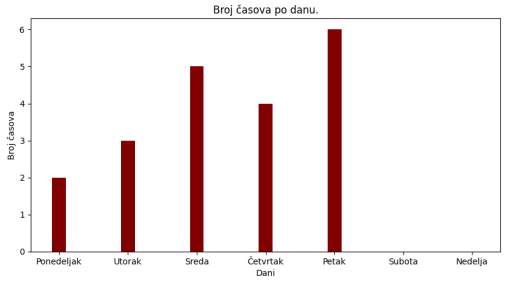
gde je <naziv_smera> IT/RN/PM i dodeljivati odgovarajući smer tom studentu.

1.6 Kalendar

Učitavanje podataka iz Google-ovog kalendara će biti omogućeno zahvaljujući postojanju JSON objekta proizvoljnog kalendara koji je nalik ugrađenim rečnicima u Python-u. Uprkos lakom učitavanju potrebno je kompleksnije parsiranje kalendara, naime jedno string polje `summary` sačinjava naziv predmeta, profesor, tip časa kao i mesto odvijanja. Smerač će to parsiranje izvesti na što efikasniji način sa minimalnim brojem petlji i korišćenjem ugrađenih i optimalnih funkcija za rad sa stringovima.

Nakon uspešno parsiranih podataka potrebno je generisati raspored prilagođen za čitanje na računarima i telefonima. Smerač će spojiti istoimene predmete ali će zapisati odvojene termine časova u zavisnoti od profesora koji ga drži kao i mesta odvijanja nastave. Primer ispisa se može videti na Slici 1. Pored tekstualnog ispisa Smerač će od istih podataka generisati stubičasti grafik koji omogućava lakši prikaz broja časova na dnevnom nivou, primer toga se nalazi na Slici 2.

Slika 1: Generisan raspored časova za sredu



Slika 2: Grafik sa dnevnim prikazom časova

2 Opis programa

2.1 Nagli prekid usled greške

Deo koda prikazan na Listingu 1 služi da prekine tok programa u potpunosti, najčešće se koristi prilikom učitavanja promenjlivih okruženja tj. u slučaju da nisu postavljene.

```
    def fail(msg):
        print(msg)
        exit(1)
```

Listing 1: Nagli prekid usled greške

2.2 Promenljive okruženja

Da bi Smerač mogao učitati potrebne postavke, koristi se biblioteka *OS*. U kodu prikazanom na Listingu 2 se takođe može videti upotreba naglog prekida kao i korišćenje nekih podrazumevanih vrednosti.

```
    def setup_config():
        config = dict()

        ...

        COMMAND = os.getenv("COMMAND")
        if COMMAND == None:
            COMMAND = "smer"
        config["command"] = COMMAND

        ...

        DISCORD_TOKEN = os.getenv("DISCORD_TOKEN")
        if DISCORD_TOKEN == None:
            fail("DISCORD_TOKEN env isn't set")
        config["discord_token"] = DISCORD_TOKEN

        NUMBER_OF_ROLES = int(os.getenv("NUMBER_OF_ROLES"))
        if NUMBER_OF_ROLES == None:
            fail("NUMBER_OF_ROLES env isn't set")

        ...

        return config
```

Listing 2: Učitavanje promenljivih okruženja

2.3 Logging

Jedna od krucijalnih funkcionalnosti svakog programa je mogućnost zapisivanja šta se dešava prilikom izvršavanja. To je najpotrebnija alatka tokom debagovanja starog ili novog koda neke naknadno dodate funkcionalnosti. Podešavanje fajla za logovanje je prikazano na Listingu 3 a opcije logovanja na Listingu 4.

```
def setup_config():

    ...

    LOG_FILE = os.getenv("LOG_FILE")
    if LOG_FILE == None:
        LOG_FILE = "/config/logs/%s.log"%(datetime.today().strftime("%Y-%m-%
    d-%H-%M-%S"))
    config["log_file"] = LOG_FILE

    ...
```

Listing 3: Podešavanje fajla za logovanje

```
def setup_logger(config):
    if os.getenv("DEBUG") == None:
        logging_level = logging.INFO
    else:
        logging_level = logging.DEBUG

    logging.basicConfig(
        filename=config["log_file"],
        level=logging_level,
        format=
        "%(asctime)s - %(name)s[%(process)s] - %(levelname)s - %(message)s",
    )
```

Listing 4: Opcije loga

2.4 Globalne promenljive

Globalne promenljive su potrebne isključivo kada više različitih funkcija zahtevaju čitanje i pisanje zajedničkih podataka. Odličan primer ovoga bi bila konfiguracija sa svim postavkama potrebnim za rad programa prikazana na Listingu 5. Pored toga, koristi se i promenljiva `log` i `bot` koja služi da interaguje sa interactions-py API.

```
log = logging.getLogger("smerac")
config = setup_config()
bot = interactions.Client(token=config["discord_token"])
```

Listing 5: Globalne promenljive

2.5 Slash komande

Slash komande se zapisuju u obliku:

```
/<naziv_komande> [opcija_1] [opcija_2] ...
```

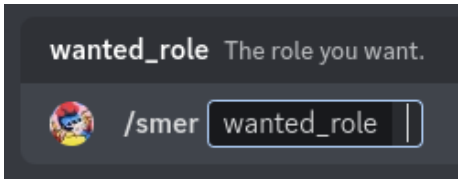
Takve komande su standardizovan način pozivanja funkcija proizvoljnog Discord bota.

2.5.1 Kreiranje slash komande

Pomoću biblioteke interactions-py moguće je kreirati slash komandu sa nazivom `smer` i obaveznom opcijom `wanted_role` što je prikazano na Listingu 6. Na Slici 3 vidljiv je izgled komande korisnicima na Discordu.

```python
@bot.command(
    name="smer",
    description="Choose your role!",
    options = [
        interactions.Option(
            name="wanted_role",
            description="The role you want.",
            type=interactions.OptionType.STRING,
            required=True,
        ),
    ],
)
```

Listing 6: Kreiranje slash komande



Slika 3: Slash komanda na Discordu

2.5.2 Dodeljivanje funkcionalnosti slash komandi

Da bi kreirana slash komanda izvršavala neki kod potrebno je odmah ispod njene kreacije deklarisati asinhrohu funkciju, prikazanu na Listingu 7, koja će izvršavati željeni kod. Ova funkcija prolazi kroz sve smerove autora komande i briše ih a potom dodaje samo željenu.

```
async def choose_role(ctx: interactions.CommandContext, wanted_role: str):
    author = ctx.author

    log.debug(f"Started choosing role for {author.nick}")

    guild = await ctx.get_guild()
    discord_roles = await guild.get_all_roles()

    for author_role_id in author.roles:
        author_role = await guild.get_role(author_role_id)
        for role in config["roles"]:
            if role.upper() == author_role.name.upper():
                await author.remove_role(author_role_id)
                log.debug(f"Removed {author.nick} from {role}")
            else:
                log.debug(f"Role {role} isn't {author_role.name}")

    if wanted_role != "none":
        for role in config["roles"]:
            if wanted_role.upper() == role.upper():
                for discord_role in discord_roles:
                    if wanted_role.upper() == discord_role.name.upper():
                        await author.add_role(discord_role.id)
                        break
                break
        log.info(f"Added {author.nick} to {wanted_role}.")
        await ctx.send(f"Succesfully added {author.nick} to {wanted_role.
    upper()}!", ephemeral=True)
    else:
        log.info(f"Removed {author.nick} from all roles.")
        await ctx.send(f"Succesfully removed {author.nick} from all roles!",
     ephemeral=True)
```

Listing 7: Dodeljivanje funkcionalnosti slash komandi

Opcija *ephemeral* znači da se poruka šalje samo autoru komande, tj. korisniku koji je pozvao komandu za izmenu smera.

2.6 Kalendar

2.6.1 Instanciranje niti

Kako bi se kalendar asinhrono učitavao za svaki od smerova odvojena je funkcija koja pronalazi sve kanale sa nazivom smera u kategoriji `calendar`, prikazana na Listingu 8, od funkcije za učitavanje, parsiranje i generisanje kalendara.

```
async def calendar(delay):
    log.debug("Calendar")

    for guild in bot.guilds:
        channels = await guild.get_all_channels()
        for category in channels:
            if category.type == interactions.ChannelType.GUILD_CATEGORY and
    category.name.upper() == "calendar":
                for role in config["roles"]:
                    for channel in channels:
                        if channel.parent_id == category.id and role.upper()
     == channel.name.upper():
                            CALENDAR_URL = os.getenv("CALENDAR_URL_" + role)
                            if CALENDAR_URL == None:
                                log.info("CALENDAR_URL_%s env isn't set,
    skipping."%(role))
                            else:
                                asyncio.create_task(updateCalendar(channel,
    CALENDAR_URL, delay))
                            break
                break
```

Listing 8: Instanciranje niti

U slučaju da ne postoji URL za kalendar proizvoljnog smera, program beleži to u log i nastavlja da se izvršava.

2.6.2 Ažuriranje kalendara

Glavna funkcija, prikazana na Listingu 9, se bavi pozivom svih pomoćnih za rad sa kalendarom. Učitava JSON objekat sa URL-a u `calendar` rečnik, generiše narednu sedmicu u `week_data` i parsira je za podatke u `week`. `week_data` i `week` su rečnici sačinjeni iz polja sa imenima svih dana u sedmici gde je svaki od njih lista događaja tj. predavanja. Na kraju vrši ispis parsirane sedmice, ako se razlikuje od prošle sedmice tj. `week_old`, i takođe šalje sliku grafa sa brojem časova po danu.

```
async def updateCalendar(channel, calendar_url, delay):
    log.debug("Updating calendar for " + channel.name)
    spacer = "-------------------------"
    week_old = dict()
    while True:
        calendar = json.loads(requests.get(calendar_url).text)
        week_data = await generateWeek(calendar["items"])
        week = await parseWeek(week_data)
        if week_old != week:
            log.debug(str(week_old))
            log.debug("----- week_old != week -----")
            log.debug(str(week))
            week_old = week
            await channel.purge(8, check=check_pinned)
            for weekday in week:
                if week[weekday] != []:
                    day_output = spacer
                    day_output += "\n\n**" + dayInWeekSrpski(weekday) + "
:**\n\n"
                    for event in week[weekday]:
                        day_output += "--- " + event["name"] + " ---\n"
                        for info in event["info"]:
                            day_output += info["name"] + "\n"
                            for i in range(len(info["start_times"])):
                                day_output += "**" + info["start_times"][i]
+ "** - " + info["end_times"][i] + "\n"
                        day_output += "\n"
                    day_output += spacer
                    await channel.send(content=day_output)
            await channel.send(files = await classesPerDayGraph(channel.name
, week))
        await asyncio.sleep(delay)
```

Listing 9: Ažuriranje kalendara

2.6.3 Generisanje sedmice

Funkcija, prikazana na Listingu 10, prolazi kroz raspored za celu godinu (JSON objekat) i uzima sve događaje koji se odvijaju u narednih 7 dana. Zatim ih svrstava po danima u rečnik `week` sačinjenen iz polja sa imenima svih dana u sedmici gde je svaki od njih lista događaja tj. predavanja i sortira događaje po početnom vremenu odvijanja. Na kraju vraća rečnik `week`.

```
async def generateWeek(events):
    log.debug("Generating week")

    current_date = date.today()
    week = {"mon": [], "tue": [], "wed": [], "thu": [], "fri": [], "sat": []
    , "sun": []}

    for event in events:
        start_date = datetime.fromisoformat(event["start"]["dateTime"]).date()
        if (start_date >= current_date) and (start_date < current_date + timedelta(days=7)):
            week[dayInWeek(start_date.weekday())].append(event)

    for weekday in week:
        week[weekday].sort(key = lambda event : datetime.fromisoformat(event["start"]["dateTime"]))

    return week
```

Listing 10: Generisanje sedmice

2.6.4 Parsiranje sedmice

Funkcija, prikazana na Listingu 11, prolazi kroz sedmicu i spaja duplikate časova u rečnik `week` koji je sačinjenen iz polja sa imenima svih dana u sedmici gde je svaki od njih lista događaja tj. predavanja, ali razlikuje ako isti čas drže različiti profesori ili se odvija na drugom mestu.

Smerač postiže to razlikovanje deljenjem polja `summary` na `name` (sve pre prve zapete) tj. naziv događaja i ostatak `info` koji predstavlja sve informacije o događaju kao što su mesto odvijanja i ime profesora ili asistenta. Nakon te podele briše sve bele praznine (eng. *white spaces*) i proverava da li u parsiranoj sedmici (rečnik `week`) već postoji događaj sa tim nazivom, ako postoji proverava se da li postoji isti `info` unutar tog događaja. Ako se naiđe na iste informacije samo se dodaju nova vremena odvijanja (početak i kraj). U slučaju da se ne pronađe već postojeći događaj ili unutar događaja informacije onda se one kreiraju.

Ovakvim parsiranjem rečnik `week` ima sledeću strukturu:

weekday	event	name	info	info name	start_times	end_times
[event]	{name, info}	'string'	{name, start_times, end_times}	'string'	['%H:%M']	['%H:%M']

'%H:%M' predstavlja string koji je konvertovan iz `datetime` ISO formata u obliku SAT : MINUT.

```
async def parseWeek(week_data):
    log.debug("Parsing week")
    week = {"mon": [], "tue": [], "wed": [], "thu": [], "fri": [], "sat": [], "sun": []}
    for weekday in week_data:
        if week_data[weekday] != []:
            for event in week_data[weekday]:
                name, info = event["summary"].split(",", 1)
                start_time = datetime.fromisoformat(event["start"]["dateTime"]).strftime("%H:%M")
                end_time = datetime.fromisoformat(event["end"]["dateTime"]).strftime("%H:%M")
                foundEvent = False
                if week[weekday] != []:
                    for event_old in week[weekday]:
                        if name.upper().replace(" ", "") == event_old["name"].upper().replace(" ", ""):
                            foundInfo = False
                            for info_old in event_old["info"]:
                                if info.upper().replace(" ", "") == info_old["name"].upper().replace(" ", ""):
                                    info_old["start_times"].append(start_time)
                                    info_old["end_times"].append(end_time)
                                    foundInfo = True
                                    break
                            if not foundInfo:
                                new_info = {"name": info, "start_times": [], "end_times": []}
                                new_info["start_times"].append(start_time)
                                new_info["end_times"].append(end_time)
                                event_old["info"].append(new_info)
                            foundEvent = True
                            break
                if not foundEvent:
                    new_event = {"name": name, "info": []}
                    new_info = {"name": info, "start_times": [], "end_times": []}
                    new_info["start_times"].append(start_time)
                    new_info["end_times"].append(end_time)
                    new_event["info"].append(new_info)
                    week[weekday].append(new_event)
    return week
```

Listing 11: Parsiranje sedmice

2.7 Kreiranje grafa

Funkcija, prikazana na Listingu 12, prolazi kroz parsiranu sedmicu i broji koliko ima časova po danu, nakon toga generiše graf uz pomoć matplot[6] biblioteke. Povratna vrednost je lokacija sačuvane slike grafa.

```
async def classesPerDayGraph(channel_name, week):
    log.debug("Plotting graph for " + channel_name)

    filename = config["saved_plots"] + "/" + channel_name + ".png"
    data = dict()

    for weekday in week:
        data[dayInWeekSrpski(weekday)] = len(week[weekday])

    keys = list(data.keys())
    values = list(data.values())

    fig = plt.figure(figsize = (10, 5))

    plt.bar(keys, values, color ='maroon', width = 0.2)

    plt.xlabel("Dani")
    plt.ylabel("Broj casova")
    plt.title("Broj casova po danu.")

    plt.savefig(filename)
    plt.close()

    return interactions.File(filename=filename)
```

Listing 12: Kreiranje grafa

3 Zaključak

Pored toga što je Smerač projekat za uspešno polaganje predmeta Skript jezici, to je i društveno korisna aplikacija koja se koristi na Discord serveru studenata Departmana za matematiku i informatiku. Najpre nudi jednostavan i bezbedan način odabira smera što smanjuje potrebu za ručnim radom neke privilegovane osobe ali i ubrzava čitav proces. Prilikom odabira smera studenti dobijaju pristup isključivo onim kanalima koji su potrebni za taj smer i time se olakšava pregled i upotreba Discord servera. Naravno, najkompleksnija funkcionalnost ovog programa jeste parsiranje kalendara, ali nakon što je ta prepreka pređena studentima je dostupan drastično brži i lakši pristup rasporedu predavanja i vežbi.

Smerač je takođe slobodnog i otvorenog koda i kao takav može biti uzor drugim aplikacija sličnog tipa, što i jeste slučaj za budući projekat jednog studenta koji planira da integriše Moodle notifikacije vezane za smer i prosledi ih na isti Discord server.

Literatura

[1] “Discord.” https://discord.com.

[2] “Python.” https://www.python.org.

[3] “interactions-py.” https://github.com/interactions-py/interactions.py.

[4] “Asyncio.” https://docs.python.org/3/library/asyncio.html.

[5] “Gpl.” https://www.gnu.org/licenses/gpl-3.0.en.html.

[6] “Matplotlib.” https://matplotlib.org/stable/tutorials/introductory/pyplot.html.