Biblioteka za Zoom API

Kovačević, Domagoj

VELEUČILIŠTE U BJELOVARU
PREDDIPLOMSKI STRUČNI STUDIJ RAČUNARSTVO

BIBLIOTEKA ZA ZOOM API

Završni rad br. 05/RAČ/2020

Domagoj Kovačević

Bjelovar, listopad 2020.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Kandidat: **Kovačević Domagoj** Datum: 31.08.2020. Matični broj: 001858

JMBAG: 0016132599

Kolegij: **C# PROGRAMIRANJE**

Naslov rada (tema): **Biblioteka za Zoom API**

Područje: **Tehničke znanosti** Polje: **Računarstvo**

Grana: **Programsko inženjerstvo**

Mentor: **Krunoslav Husak, dipl.ing.rač.** zvanje: **predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Ivan Sekovanić, mag.ing.inf.et comm.techn., predsjednik**
2. **Krunoslav Husak, dipl.ing.rač., mentor**
3. **Tomislav Adamović, mag.ing.el., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 05/RAČ/2020

U radu je potrebno u C# programskom jeziku izraditi biblioteku koja omogućava rad s programskim sučeljem Zoom platforme.

Cilj završnog rada je izraditi dovoljan broj klasa koje bi koristeći C# programski jezik omogućavale upravljanje sastancima na Zoom platformi.

Biblioteka će omogućiti daljnji razvoj desktop i web aplikacija koje će olakšati korištenje Zoom platforme na Veleučilištu u Bjelovaru.

Zadatak uručen: 31.08.2020. Mentor: **Krunoslav Husak, dipl.ing.rač.**

Zahvala

Zahvaljujem Krunoslavu Husaku, dipl.ing.rač. na mentorstvu, ne samo tijekom završnog rada već tijekom sveukupnog fakultetskog obrazovanja te svim profesorima Veleučilišta u Bjelovaru što su mi priuštili najbolje moguće studentsko iskustvo i bili podrška tijekom studiranja.

Sadržaj

1. UVOD

U današnje vrijeme i s obzirom na epidemiološke okolnosti potreba za socijalnim razmakom je sve veća. Nastava na daljinu je pojam naše današnjice i sve više škola, fakulteta i korporacija koriste servise koji im to dozvoljavaju. Jedan od popularnijih servisa je Zoom o kojemu ćemo u ovome radu govoriti. Ovaj završni rad bazira se na izradi C# biblioteke za Zoom API. U radu je pobliže opisano kreiranje aplikacije koja služi za prikaz nekih krajnjih točaka *(eng. endpoint)* Zoom-ovog API-a. Rad je napravljen u svrhu lakšeg korištenja Zoom-ovih funkcionalnosti te kako bi imali sve funkcionalnosti na jednom mjestu.

U drugom poglavlju opisano je okruženje koje se koristilo u svrhu izrade rada, dok je u trećem opisana Zoom platforma. Zatim slijedi API koji se spominje u gotovo cijelom radu, potom Biblioteka za Zoom API. Od petog poglavlja govorimo o kreiranju rada te programiranju i testiranju različitih funkcionalnosti.

Konačan rezultat je aplikacija kojom možemo dohvaćati, brisati, kreirati sastanke i korisnike bez potrebe da uključimo Zoom.

2. OKRUŽENJE

Okruženje je gotovo svaki program instaliran na računalu sa zadaćom da služi za razvoj ili testiranje aplikacija. Normalno programsko okuženje sastoji se od uređivača izvornog koda, ugrađenih alata za automatizaciju i programa za ispravljanje grešaka *(eng. debugger)* [1]. Za ovaj rad korišteno je programsko okruženje Visual Studio 2019.

2.1 Visual Studio 2019.

Visual Studio je integrirano programsko okruženje *(eng. Integrated development environment)* korišteno za razvoj računalnih programa na sustavu Windows. Također se koristi za izradu web stranica, web aplikacija i web servisa te koristi platforme za razvoj poput Windows Forms, Windows API, Microsoft Silverlight... Uključujući uređivač koda *(eng. code editor)* koji podržava IntelliSense, Visual Studio je napisan kompletno u C++-u i C#-u te nudi ugrađen program za otklanjanje pogrešaka koji otklanja i greške u izvornom kodu i one na strojnom dijelu. Visual Studio podržava trideset šest različitih programskih jezika od kojih su samo neki: C, C++, Visual Basic, .NET, C#, JavaScript, HTML, CSS. Podrška za jezike kao što su Ruby, Node.js i Python moguća je preko priključaka *(eng. plug-in)* [2].

2.2 Programski jezik C#

C# je programski jezik koji se pokreće u .NET okviru *(eng. framework)*. Razvijen je od strane Microsofta 2001. godine. Svrha C#-a bila je razviti programski jezik koji nije samo lagan za naučiti, već podržava suvremene funkcionalnosti za sve vrste softverskog razvoja. Jednostavan je, suvremen i objektno orijentiran programski jezik koji dostavlja modernim programerima fleksibilnost i značajke za razvoj programa koji neće samo danas raditi već će se moći primjenjivati godinama. Windows nije jedina platforma na kojoj se C# može koristiti, .NET aplikacije mogu se razvijati i na Linux-u i Mac-u. Mnogi bi nazvali C# švicarskim nožem programiranja jer njime možemo izraditi Windows klijentske aplikacije, komponente i biblioteke, servise, aplikacije za mobilne uređaje i video igre... Jedna od bitnijih informacija vezanih za C# je da on kao programski jezik raste i ako pogledamo kroz povijest, C# je jedan od programskih jezika koji najbrži evoluira zahvaljujući stalnoj podršci od Microsofta i zajednice. Originalno je dizajniran za pisanje Windows klijentskih aplikacija, no u C#-u danas možemo poprilično sve uključujući pisanje konzolnih aplikacija, cloud aplikacija i modernih programa za strojno učenje *(eng. machine learning)* [3].

2.3 Windows Forms

Windows obrasci ili *Windows Forms* su grafičke biblioteke otvorenog koda *(eng. open-source graphical class library)* koje su dio .NET okvira ili Mono okvira. One su platforma za pisanje programskog koda klijentskih aplikacija za laptope, tablete i PC-e. U ovom radu korišteni su *Windows Forms* obrasci za testiranje te prikaz podataka, kao i ugodnije korisničko iskustvo [4].

2.3.1 ***Detaljnije***

Windows Forms ili Windows obrasci su set upravljanih biblioteka koje pojednostavljuju aplikacijske zadatke kao što su čitanje i pisanje u datotečni sustav. Kada koristimo okruženje kao što je Visual Studio možemo kreirati pametne klijentske aplikacije za prikaz informacija i zahtjeve od korisnika te komunicirati s računalima preko mreže. U Windows obrascima, jedan obrazac *(eng. form)* je vidljiva podloga na kojoj korisniku prikazujemo informacije [5].

2.3.2 ***Kontrole***

Obrascima su obično potrebne kontrole koje dodajemo i čiju funkcionalnost programiramo, kao što su akcije nad gumbovima, klikovi miša ili pritisci tipkovnice. Kontrola je UI *(eng. user interface)* element koji prikazuje podatke ili prihvaća unos podataka [5]. Kada god korisnik napravi neku akciju, ona generira događaj *(eng. event)*. Aplikacija reagira na te događaje koristeći kod i procese kada se događaji izvrše ili „dogode“. Windows obrasci donose nam paletu kontrola koje možemo dodati na obrazac od kojih su samo neke kontrole za prikaz polja za unos teksta, gumbova i padajućih izbornika.

2.3.3 ***Arhitektura***

Windows Forms aplikacija je događajem potaknuta aplikacija, za razliku od *.bat* programa, koristi većinu svog vremena čekajući korisnika da nešto učini, kao npr. popuni polje za tekst ili klikne na gumb.

2.3.4 ***Značajke***

Svi vidljivi elementi u *Windows Forms* biblioteci klasa potiču iz kontrolne ili „Control“ klase. Time se pospješuje minimalna funkcionalnost UI elemenata kao što su lokacija, boja, tekst, font isto kao i uobičajeni događaji npr. klik [5].

3. ZOOM

3.1 Općenito

Zoom je sustav za video konferencije baziran na oblaku *(eng. cloud)*. Osnovao ga je 2011. godine Eric Yuan sa zadaćom da sustav video konferenciranja bude lakši i pristupačniji. Javno je dostupan od 2013. godine [6]. Svrha sustava je virtualan susret s ljudima bilo to preko kamere, zvuka ili istovremeno jednog i drugog, i to sve uz čavrljanje u realnom vremenu *(eng.chat)*. Također nam daje mogućnost da snimimo sjednicu *(eng. session)* kako bismo je gledali kasnije.

3.1.1 *Zoom sastanak*

Obično čujemo ljude kako govore Zoom sastanak *(eng. meeting)* ili Zoom soba *(eng. room)*, u tom slučaju sastanak je video konferencija koju je napravio domaćin *(eng. host)* preko servisa, a soba je fizičko sklopovlje koje dozvoljava tvrtkama zakazivanje i pokretanje Zoom sastanaka iz njihovih konferencijskih prostorija [7]. Zoom sobe zahtijevaju dodatnu pretplatu na već postojeću Zoom pretplatu i one su idealna solucija za veće kompanije.

3.1.2 *Detaljnije o Zoom-u*

Tajna Zoom-ove popularnosti leži u jednostavnosti korištenja platforme. Napraviti Zoom konferenciju možemo u tri jednostavna koraka: napravimo Zoom račun, posjedujemo web kameru i pristup internetu. Zoom također sugerira korisnicima da se služe njihovom aplikacijom za radnu površinu *(eng. desktop application)*. Mobilni korisnici moraju preuzeti aplikaciju za svoj operativni sustav kako bi prisustvovali sastanku. Platforma je kompatibilna sa sustavima Windows, Mac, Linux te iOS i Android [6]. Uspoređujući Zoom s konkurentskim tvrtkama, i dalje nama korisnicima nudi jednu od najboljih besplatnih i laganih opcija za stvaranje video konferencija te sigurnost prilikom izvođenja istih.

3.2 Kako koristiti Zoom?

Kako bismo napravili sastanak moramo otići na Zoom-ovu web stranicu na kojoj ako smo registrirani korisnik možemo odmah pritisnuti *Host a meeting*, prilikom čega nam se prezentira opcija da započnemo sastanak s videozapisom ili bez istog. Možemo također napraviti sastanak zakazan za neki drugi datum ili vrijeme, prilikom čega nam se prezentira obrazac na kojem ispunjavamo opcije kao što su: uključivanje i isključivanje kamera,

snimanje sastanka, opcionalna soba za čekanje itd. Kada smo kreirali sastanak možemo ga odmah podijeliti putem Google, Outlook i Yahoo kalendara, popunimo polja za e-mail adrese sudionika koje želimo pozvati i oni će elektronski dobiti poziv na isti. Unutar sastanka imamo također mnoštvo opcija koje se mogu koristiti generalno ili nad nekim korisnikom ako smo domaćin sastanka, neke od njih su: ugasiti zvučni zapis sudionika, ugasiti video sudionika, izbaciti sudionika iz sastanka, dozvoljavanje sudioničkih web kamera itd [6].

3.3 Značajke

Tri glavne značajke Zoom servisa su:

- Jedan na jedan sastanci – moguće je održati beskonačno sastanaka s besplatnim korisničkim planom
- Grupne video konferencije – održite konferenciju s do petsto sudionika (u slučaju kupovine *large meeting* dodatka). Besplatni plan nam dozvoljava kreiranje video konferencije s do sto sudionika.
- Dijeljenje ekrana *(eng. screen sharing)* – jedan na jedan ili grupno dijeljenje ekrana sa sudionicima sastanka

3.4 *Kako radi Zoom?*

Zoom dozvoljava jedan na jedan sastanke koji mogu prerasti u grupne pozive i web seminare te globalne video sastanke s do tisuću sudionika i četrdeset devet videa na ekranu. Besplatni plan dozvoljava beskonačno jedan na jedan sastanaka ali limitira grupne sastanke na trajanje od četrdeset minuta i do sto sudionika. Cijena osnovnog plaćenog plana je petnaest dolara. Ostali Zoom pretplatni planovi su:

- Zoom Pro – dozvoljava kreiranje osobnog id-a sastanka *(eng. meeting ID)*, te dozvoljava snimanje sastanaka
- Zoom Business – dozvoljava brendiranje sastanaka, te prijepis istih *(eng. transcript)* Zoom sastanka
- Zoom Enterprise – beskonačan prostor za pohranu videozapisa sastanaka
- Zoom Rooms

3.5 Zoom API

Zoom API dozvoljava programerima da na siguran način pristupe informacijama od Zoom-a. Njihov API koristi se u svrhu izrada privatnih servisa ili javnih aplikacija na Zoom tržnici *(eng. Zoom Marketplace)*. Kako bi naučili kako se koristi Zoom API, Zoom

je napravio web stranicu na kojoj su prikazane sve krajnje točke *(eng. endpoint)* dostupne preko HTTPS protokola. Npr. ako želimo naučiti kako dohvatiti sve korisnike jednog korisničkog računa na Zoom-u to možemo učiniti preko web stranice: https://api.zoom.us/v2/users [8].

3.6 Zoom tržnica

Zoom tržnica *(eng. Zoom App Marketplace)* je sigurna platforma otvorenog koda koja dozvoljava programerima izradu aplikacija koje koriste Zoom-ovu komunikacijsku platformu (za razgovor, video konferencije i sl.). Od svog osnutka u 2018. godini, Zoom tržnica nastavlja rasti s desecima tisuća programera zaposlenih na izradi proizvoda koji rade milijune sigurnih API poziva svaki mjesec [9]. Na Zoom tržnici možemo napraviti nekolicinu aplikacija pomoću njihovog API-a, kao što su: JWT, *OAuth*, *Chatbot* itd.

3.6.1 ***Javne i privatne aplikacije***

Aplikacije izlistane na Zoom tržnici su sve **Javne** aplikacije. Kreirali su ih programeri koji proširuju projekt na Zoom platformu kao implementaciju. One uključuju aplikacije kao što je Hugo [9]. **Privatne** aplikacije nisu izlistane na direktoriju tržnice. One su dodatak Zoom platformi koji pomaže tvrtkama kreirati inovativne alate koji će npr. kreirati korisnike za korisnički račun tvrtke na Zoom-u, dohvaćati sve sastanke, ažurirati podatke o sastancima i sl.

3.6.2 ***Privatnost i sigurnost***

Svaka aplikacija poslana javno na Zoom tržnicu prolazi kroz temeljit proces pregledavanja kako bi se osigurala kvaliteta i sigurnost. U trenutku kad je aplikacija poslana, tim programera Zoom tržnice obavlja sigurnosnu procjenu i penetracijske testove čija je zadaća osigurati i zaštititi korisničke podatke.

3.6.3 ***JWT***

JWT *(eng. JSON Web Tokens)* su linije teksta poslane na svaki zahtjev ne bi li se potvrdila autorizacija od strane servera. One sadrže kombinaciju tajni *(eng. secrets)* od API-a i korisne nosivosti *(eng. payload)* u obliku JSON objekta. U ovom završnom radu kreirana je JWT aplikacija u svrhu komunikacije sa Zoom serverom te korištenja podataka sa Zoom-a [10].

3.6.4 ***Aktivacija aplikacije***

Kada možemo pristupiti poljima pod nazivom *App Credentials*, aplikacija je spremna za slanje zahtjeva na Zoom API. Također imamo opciju *Deactivate* ili *Reactivate*, ako pritisnemo prvu opciju nismo više autorizirani slati zahtjeve na Zoom API.

4. API

4.1 *Općenito*

API *(eng. application programming interface)* je računalno sučelje koje definira postupke između više softverskih posrednika. Definira vrste poziva ili zahtjeva koji su mogući, kako ih kreirati, formate u kojima bi podaci koji se šalju trebali biti, pretvorbe koje slijede i sl. Također nam može pružati mehanizme za proširenje koje korisnici na već postojećim funkcionalnostima mogu iskoristiti na različite načine [11]. Za API-e se nekad smatra da su ugovori s dokumentacijom koja prezentira dogovor između dvije stranke. Ako prva stranka pošalje zahtjev na određen način, točno je određen način na koji druga stranka odgovara. Zato što API-i olakšavaju integraciju novih elemenata u postojeću arhitekturu, oni pomažu timovima, IT tvrtkama i poslovanjima diljem svijeta. Kako bi se pobliže opisalo što je API, u svrhu ovog projekta kreiran je dijagram koji vidimo na slici 4.1.

4.1.1 ***Web servisi***

Internetski *(eng. web)* servis je resurs koji je dostupan preko interneta, iz tog razloga mu je potrebna mreža. Termin web servis koristi se od 2000. godine, i to je zapravo specifična vrsta otvorenog API-a. Namijenjeni su tako da se koriste kao dio servisno orijentirane arhitekture *(eng. service-oriented architecture)* korištene za dizajniranje softverskih aplikacija gdje su funkcionalnosti predstavljene kao servisi jedne mreže. U suštini, web servis je resurs koji obavlja specifičan zadatak [16].

4.1.2 ***Web servisi protiv API-a***

Svaki web servis je API, zato što otkriva podatke ili aplikacije, ali nije svaki API web servis. To je iz razloga što u usporedbi s API-ima, web servisima je potrebna mreža. Dok API-i mogu koristiti bilo koji protokol ili stil dizajna, web servisi obično koriste SOAP (nekada REST, UDDI i XML-RPC) [16].

4.2 *Svrha*

U izradi programskih aplikacija, API pojednostavljuje programiranje oduzimajući implementaciju i otkrivajući samo objekte ili akcije koje su potrebne programeru. Gdje grafičko sučelje za e-mail možda opskrbi svog korisnika tipkom koji napravi sve korake za prikupljanje i ispis e-mailova, API za unos i/ili ispis podataka može dati programeru funkciju koja kopira datoteku s jedne lokacije na drugu bez da programer razumije operacije koje će se izvršiti „iza kulisa“ [11].

4.3 *API primjeri*

Postoji mnogo primjera API-a koji su javno dostupni i s kojima možete lako doći u susret, mnoge od kojih su napravili giganti industrije. Mogućnost pristupa programskom kodu platforme neke tvrtke i to programski preko API-a je ono što ih čini platformama [12]. Neki od primjera API-a:

- Google API-i – dozvoljavaju spajanje našeg koda s nekolicinom Google-ovih servisa kao što su *Maps* i *Translate*. Google-u su API-i toliko bitni da koriste Apigee, vodeću platformu za upravljanje API-ima.
- Facebook API-i – dozvoljavaju da programski pristupimo Facebook-ovim socijalnim grafovima i marketinškim alatima
- Twitter API
- Java API
- Zoom API – kao što je navedeno u poglavlju 3.5

4.3.1 ***Google API-i***

Google API-i dostavljaju funkcionalnosti poput analize podataka i strojnog učenja *(eng. machine learning)* kao servis ili za pristup korisničkim podacima. Postoji puno klijentskih biblioteka u različitim programskim jezicima koje dozvoljavaju korisnicima korištenje Google API-a u kodu uključujući jezike kao što su: Java, JavaScript, Ruby, .NET.

4.3.2 ***Twitter API***

Po definiciji, Twitter API je JSON API koji dozvoljava programerima programski pristup Twitter informacijama. Twitter API-u isto kao i Zoom API-u, moramo preko interneta pristupiti zahtjevima čiji je domaćin *(eng. host)* Twitter. Jedan od osnovnih elemenata Twittera je *tweet*. Twitter API govori nam što možemo raditi s tim objavama, kao npr. izraditi novu, pretražiti objave i sl., isto kao i kako putem programa doći do tih mogućnosti [12].

4.3.3 ***Java API***

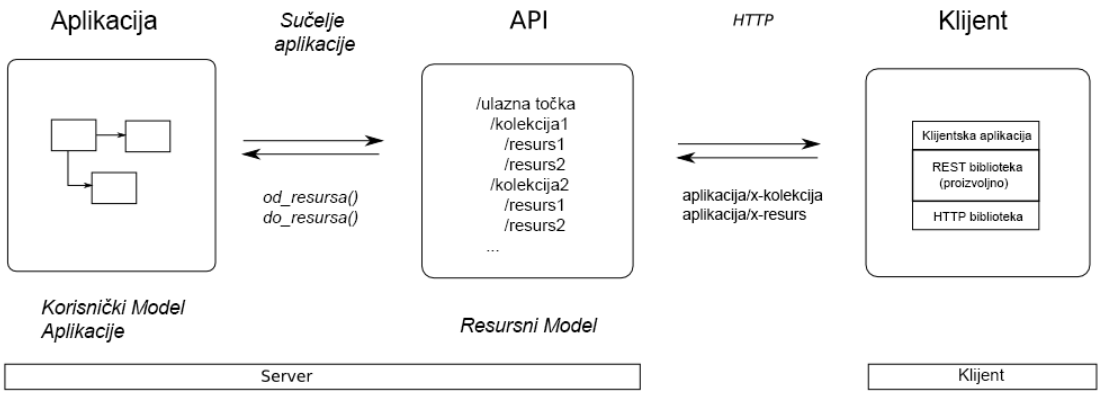
Java API je biblioteka softverskih komponenti dostupnih odmah bilo kome tko ima instaliran JDK *(eng Java Development Kit)*. Te komponente implementiraju obične zadatke i generalno povećavaju produktivnost zato što programeri ne moraju svakog puta počinjati pisati kod iz početka. Jedna od osnovnih komponenata korištena u softveru je

lista. Java API određuje što možemo učiniti s listom (dodati elemente, sortirati elemente itd.) [12].

4.4 *Moderan API*

Tijekom godina, API je generalno opisan kao bilo kakva poveznica između sučelja i aplikacije [12]. U današnje vrijeme moderan API poprimio je karakteristike koje ga čine ekstremno vrijednim i korisnim kao što su:

- Podržavanje HTTP i REST standarda, koji su lako dostupni i opće shvaćeni standardi.
- Prema API-ima se ponaša više kao prema proizvodu. Dizajnirani su za specifične klijente kao npr. mobilne programere, dokumentirani su i verzionirani tako da korisnici mogu imati uvid u to kako radi API.
- Zato što su više standardizirani, imaju jaču sigurnost te su praćeni kako bi se pojačale sposobnosti API-a.
- Moderan API sadrži svoj vlastiti životni ciklus softverskog razvoja *(eng software development lifecycle*) ili SDLC za dizajniranje, testiranje, izgradnju i verzioniranje.



Slika 4.1: Dizajn API-a

5. BIBLIOTEKA ZA ZOOM API

5.1 Općenito o bibliotekama

Biblioteka u programiranju je kolekcija obrađenih i neuništivih rutina koje koristi program. Rutine, koje se nekada nazivaju modulima, mogu sadržavati konfiguracijske podatke, dokumentaciju, predloške poruka, podrutine, klase i vrijednosti [13].

5.1.1 *Povijest*

Najraniji koncepti programiranja bili su analogni bibliotekama i namijenjeni razdjeljivanju podataka na definicije kako bi se lakše implementirale. Godine 1965., programski jezik Simula koristi klase koje su mogle biti uključene u biblioteke i dodane prilikom stvaranja programa [14].

5.1.2 *Razlog za korištenje biblioteka*

Najbolji „adut“ biblioteka i razlog za njihovom uporabom je ponovna iskoristivost njihovog ponašanja. Kada koristimo biblioteku, program poprima ponašanje implementirano unutar te biblioteke bez potrebe za ručnom implementacijom. Korištenje biblioteke jednako je korištenju individualnih objekata u njoj ali je puno jednostavnije što samo moramo imati posla s jednom datotekom u usporedbi s više njih ili više stotina njih od kojih je ona izgrađena.

5.1.3 *Vrste biblioteka*

Implementacija je specifična za svaki programski jezik ili sustav. Ako stvaramo npr. C ili C++ program na Windowsu, imamo neke opcije [15]:

- Statična biblioteka *(eng. static library)*
- Dinamična povezana biblioteka *(eng. dynamic linked library)*
- Dijeljena biblioteka *(eng. shared library)*

5.2 *Kreiranje JWT aplikacije*

U svrhu izrade završnog rada, potrebna je bila kreacija JWT aplikacije. Tijekom sljedećih nekoliko potpoglavlja bit će opisani koraci za kreiranje JWT aplikacije.

5.2.1 *Navigacija*

Kako bismo izradili našu JWT aplikaciju navigiramo se na web stranicu Zoom tržnice kao što je navedeno u poglavlju 3.6. Na stranici pritiskom na *Sign In* dovedeni smo na obrazac koji ispunjavamo našim Zoom korisničkim podacima. Ukoliko smo već

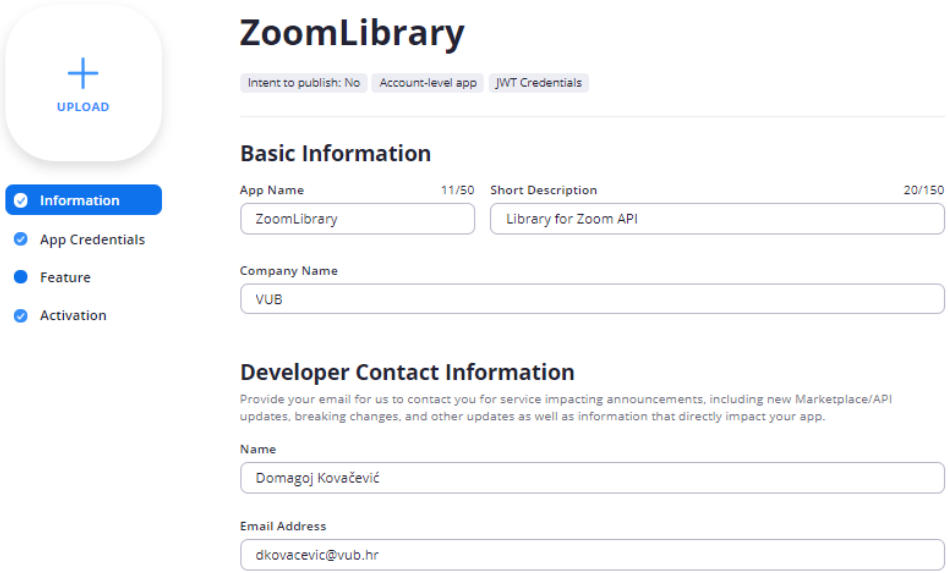
prijavljeni na glavnoj Zoom web stranici, *Sign In* nas direktno prijavljuje na tržnicu. Pritiskom na *Develop* padajući izbornik, dobivamo pristup opcijama kao što su: *Build App, Documentation*, *Programmer Blog* i *Community Forum*.

5.2.2 ***Sagradi aplikaciju***

Sagradi aplikaciju *(eng build application)* ili *Build App* odjeljak padajućeg izbornika iz prošlog potpoglavlja vodi nas na web stranicu Zoom tržnice s mnoštvom opcija za kreiranje različitih tipova aplikacija kao što je navedeno u 3.6. Pritiskom na *Create* kod odjeljka JWT kreirali smo JWT aplikaciju, sada možemo pristupiti informacijama te aplikacije te ključnim faktorima u ovom završnom radu kao što su API ključ *(eng. API Key)* i API tajna *(eng. API Secret)* kako bi se kreirao JWT.

5.2.3 ***Informacije***

Na stranici informacije moramo popuniti polja pod naslovima osnovne informacije *(eng. basic information)* i kontaktne informacije programera *(eng. developer contact information)* kao što je u svrhu ovog rada prikazano na slici 5.1.

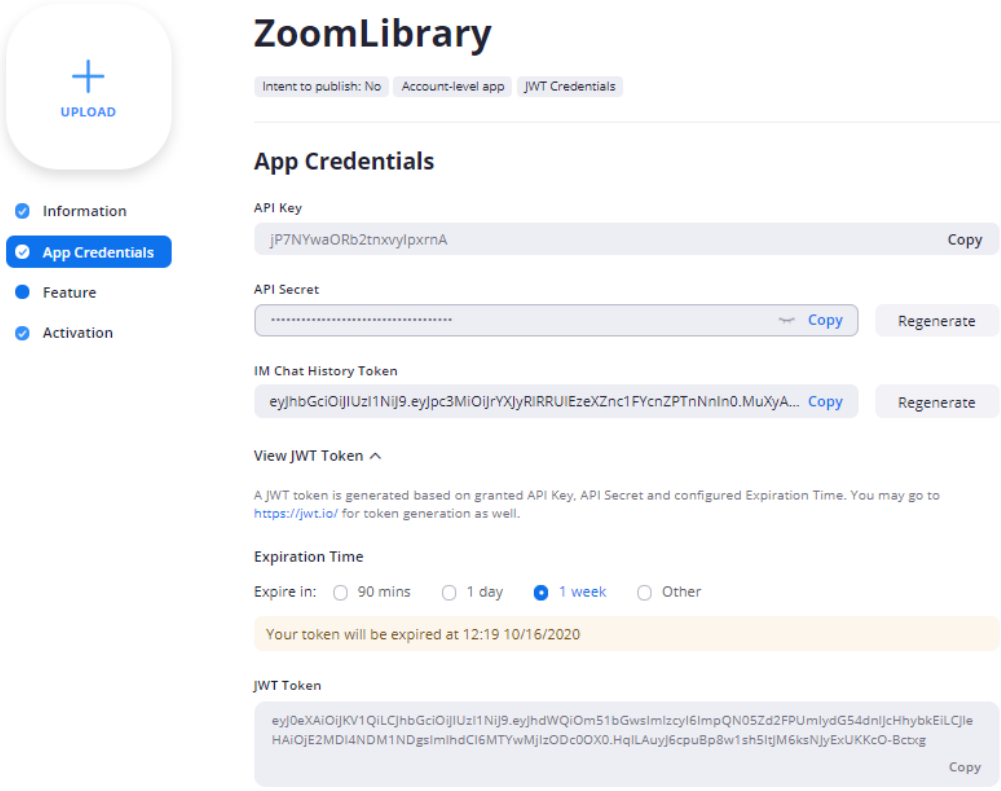


Slika 5.1: Informacije JWT aplikacije

5.2.4 ***JWT podaci***

Na stranici za JWT podatke vidimo polja API ključ i API tajna te ih klikom miša možemo kopirati. Pritiskom na *View JWT Token* prikazuje se odjeljak stranice u kojem se generira žeton. U polju rok trajanja *(eng. expiration date)* možemo odabrati koliko će dugo

žeton trajati. Sam žeton je izgeneriran ispod roka trajanja te se također može kopirati klikom miša kao što je prikazano na slici 5.2.



Slika 5.2: Podaci JWT aplikacije

5.2.5 *Aktivacija*

Pritiskom na *Activation* odjeljak našoj aplikaciji je dozvoljeno pozivati bilo koju krajnju točku. U bilo kojem trenutku se na ovu stranicu možemo vratiti i deaktivirati aplikaciju, međutim tad ona neće moći pozivati API.

5.3 *API rukovatelj*

U svrhu spajanja programskog jezika C# i Zoom API-a kreirana je klasa pod nazivom API rukovatelj *(eng. API handler)*. Unutar klase nalaze se metode za slanje poziva i zahtjeva na Zoom API, koje su u sljedećim potpoglavljima detaljnije pojašnjene. Iznad tih metoda definirane su varijable potrebne za izvršavanje metoda kao što je prikazano na slici 5.3. Svaka od ovih metoda sadrži provjere za uhvaćene krajnje točke te su sve pogreške pohranjene preko zapisivača *(eng. logger)* o kojemu će se govoriti kasnije u radu.

```
68 references
public class APIHandler : IAPIHandler
{

    public static HttpClient client = new HttpClient();

    4 references
    public static string zoomApiBaseUrl { get; set; }

    8 references
    public static string zoomApiUrl { get; set; }

    6 references
    public static string userId { get; set; }

    8 references
    public static string jwtToken { get; set; }

    0 references
    public static string endpoint { get; set; }
```

Slika 5.3: Varijable API rukovatelja

5.3.1 ***Dohvati***

Dohvati *(eng. get)* metoda služi nam kako bi dohvatili podatke sa Zoom API-a kao npr. prikazivanje liste svih sastanaka koji su zakazani za korisnika. Funkcija prima dva parametra, kao što je prikazano na slikama 5.3.1 i 5.3.2. Parametar krajnje točke *(eng. endpoint)* je ključan, s obzirom na to da se koristi u više metoda i svaki put je različit ovisno o tome koju funkcionalnost API-a želimo, ako npr. želimo promijeniti informaciju preko zahtjeva, tada se neće koristiti *Get* već *Delete* metoda te će nam tada trebati druga krajnja točka. Na slici 5.4 nalazi se dio izvornog koda potreban za pristup Zoom API-u, dok slika 5.5 prikazuje pozivanje *GetAsync* metode kojoj prosljeđujemo *zoomApiUrl* i *endpoint* varijable.

5.3.2 ***Postavi***

Postavi *(eng. post)* metoda slična je kao i *Get* samo što njome ne dohvaćamo podatke, već ih šaljemo. *Post* metoda prima krajnju točku i HTTP tijelo, a unutar metode nalazi se provjera da li su podaci uspješno poslani na API ili ne. Ova metoda služi nam kako bismo npr. zakazali sastanak ili kreirali korisnika za postojeći Zoom račun.

5.3.3 *Obriši*

Obriši *(eng. delete)* metoda kao i prethodne, prima parametar krajnje točke koji joj služi za pristup Zoom API-u za primjerice brisanje korisnika.

5.3.4 *Popravi*

Popravi *(eng. patch)* metoda također prima krajnju točku i HTTP tijelo, no međutim prima i opcionalni *query* parametar koji nam služi kako bi promijenili korisničke informacije nekog korisnika, ili promijenili informacije postojećeg sastanka.

5.3.5 *Umetni*

Umetni *(eng. put)* metoda nam dozvoljava da pristupimo jednom sastanku i prekinemo ga izravno, ili ga ako je obrisan vratimo i ponovo kreiramo.

```
4 references
public string Get(string endpoint, HttpClient httpClient = null)
{

    string retVal = "";

    if (httpClient == null)
    {
        APIHandler.client = new HttpClient();
        APIHandler.client.DefaultRequestHeaders.Add("Accept", "application/json, application/xml");
        APIHandler.client.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", APIHandler.jwtToken);
    }
    else
    {
        APIHandler.client = httpClient;
    }
```

Slika 5.4: Primjer izvornog koda za metodu Get

```
try
{
    HttpResponseMessage response = APIHandler.client.GetAsync($"{APIHandler.zoomApiUrl}/{endpoint}").GetAwaiter().GetResult();
    if (response.IsSuccessStatusCode)
    {
        string responseBody = response.Content.ReadAsStringAsync().GetAwaiter().GetResult();
        Console.WriteLine(responseBody);
        retVal = responseBody;
    }
    else
    {
        string responseBody = response.Content.ReadAsStringAsync().GetAwaiter().GetResult();
        throw new Exception(responseBody);
    }
    return retVal;

}
catch (HttpRequestException ex)
{
    Console.WriteLine("\nException on /APIHandler/Get endpoint method caught!!");
    Console.WriteLine("Message: {0}", ex.Message);
}

return retVal;
```

Slika 5.5: Primjer drugog dijela izvornog koda za metodu Get

5.4 *Zoom žeton*

Zoom žeton *(eng. token)* klasa služi za kreaciju JWT navedenog u potpoglavlju 5.2.4. Sastoji se od privatnih varijabli potrebnih za kreaciju te metode za samo kreiranje žetona. Metoda kalkulira vrijeme koliko će trajati žeton, a prima parametre API ključ i API tajnu. Klasa također sadrži konstruktor *Token* kojeg smo iskoristili kako bi napravili obrazac za prijavu u aplikaciju. Kao što vidimo na slici 5.6, ključ moramo osigurati preko sigurnosnih algoritama kako bi se informacije „potpisale“. Objekt korisne nosivosti stvaramo i punimo u ovom dijelu kao što je vidljivo na slici 5.7. Ne smijemo zaboraviti uključiti *Microsoft.IdentityModel.Tokens* biblioteku kako bi mogli koristiti klasu *JwtPayLoad*.

```
3 references
public class ZoomToken
{
    private JwtPayload payload;
    private string tokenString;

    1 reference
    public ZoomToken(string ZoomApiKey, string ZoomApiSecret, long Minutes)
    {
        DateTime Expiry = DateTime.UtcNow.AddMinutes(Minutes);
        string ApiKey = ZoomApiKey;
        string ApiSecret = ZoomApiSecret;

        int ts = (int)(Expiry - new DateTime(1970, 1, 1)).TotalSeconds;

        // Create Security key using private key:
        var securityKey = new Microsoft.IdentityModel.Tokens.SymmetricSecurityKey(Encoding.UTF8.GetBytes(ApiSecret));

        var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);
```

Slika 5.6: Primjer izvornog koda za klasu ZoomToken

```
        //Finally create a Token
        var header = new JwtHeader(credentials);
        var payload = new JwtPayload();

        //Zoom Required Payload
        payload = new JwtPayload()
            {
                { "iss", ApiKey},
                { "exp", ts },
            };

        var secToken = new JwtSecurityToken(header, payload);
        var handler = new JwtSecurityTokenHandler();

        // Token to String so you can use it in your client
        tokenString = handler.WriteToken(secToken);

        //string Token = tokenString;
        this.Token = tokenString;

    }

    2 references
    public string Token { get; set; }
}
```

Slika 5.7: Primjer drugog dijela izvornog koda za klasu ZoomToken

5.5 *Zapisivač*

Zapisivač *(eng. logger)* je grupacija klasa stvorenih s razlogom da se pojednostavi rukovanje s pogreškama te da se iste zapisuju u datoteke, i spremaju s odgovarajućim datumom i vremenom.

5.5.1 *Detaljnije o zapisivačima*

U svrhu pojednostavljenja programa, zapisivači su dodani kako bi se mogli pozvati na svakom mjestu u programu gdje su potrebni, kao npr. kada neki zahtjev ne uspije proći.

5.5.2 *Baza zapisivača*

Baza zapisivača ili *LogBase* klasa služi nam za kreiranje putanje u kojoj će se datoteka stvoriti te unos podataka koji će biti uneseni u datoteku. Kao što je vidljivo iz slike 5.8, baza je programirana na način da se stvori datoteka *zoom_api_library.log* u direktoriju gdje se pokreće program.

```
namespace zavrsni_rad
{
    2 references
    public abstract class LogBase
    {
        protected readonly object lockObj = new object();
        2 references
        public abstract void Log(string message);
    }

    6 references
    public class FileLogger : LogBase
    {
        private static string fileName = "zoom_api_library";

        private DirectoryInfo AddOnDirectory = Directory.CreateDirectory($"{AppDomain.CurrentDomain.BaseDirectory}Logs");

        public static string filePath = $"{AppDomain.CurrentDomain.BaseDirectory}Logs\\{fileName}.log";
        2 references
        public override void Log(string message)
        {
            lock (lockObj)
            {
                using (StreamWriter streamWriter = new StreamWriter(filePath, true))
                {
                    streamWriter.WriteLine(DateTime.Now + " " + message);
                    streamWriter.Close();
                }
            }
        }
    }
```

Slika 5.8: Primjer izvornog koda za bazu zapisivača

5.5.3 *Meta zapisa*

Meta zapisa ili *LogTarget* je klasa u kojoj određujemo tipove zapisa povezane s evidentiranjem grešaka. Kao što je vidljivo iz slike 5.9, to nije nužno samo datoteka.

```
15 references
public enum LogTarget
{
    File, Database, EventLog
}
```

Slika 5.9: Primjer izvornog koda za metu zapisa

5.5.4 *Pomoćnik zapisivača*

Pomoćnik zapisivača ili *LogHelper* je klasa koja kao što joj i ime govori, pomaže baznoj klasi te sadrži metodu zapiši *(eng. log)* koja prima lokaciju na koju spremamo zapis i poruku koja će se unutra zapisati kao što je vidljivo iz slike 5.10.

```
13 references
public static class LogHelper
{
    private static LogBase logger = null;
    13 references
    public static void Log(LogTarget target, string message)
    {
        switch (target)
        {
            case LogTarget.File:
                logger = new FileLogger();
                logger.Log(message);
                break;
            //case LogTarget.Database:
            //    logger = new DBLogger();
            //    logger.Log(message);
            //    break;
            //case LogTarget.EventLog:
            //    logger = new EventLogger();
            //    logger.Log(message);
            //    break;
            default:
                return;
        }
    }
}
```

Slika 5.10: Primjer izvornog koda za pomoćnik zapisivača

5.5.5 *Poziv u programu*

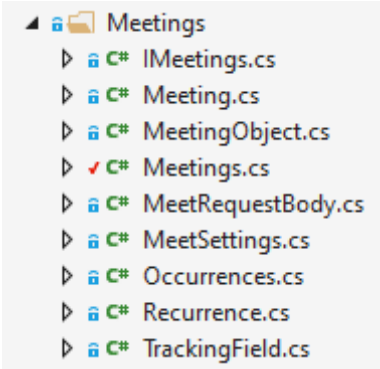
Kako bismo pozvali zapisivač u programu moramo imati pokušaj-uhvatiti dio *(eng. try-catch)*, u *catch* dijelu programskog koda trebamo proslijediti putanju za zapis i pozvati pomoćnik kako bi se greška zapisala u datoteku kao što je vidljivo iz slike 5.11 te kako bismo uhvatili grešku.

```
catch (Exception ex)
{
    Console.WriteLine(FileLogger.filePath);
    LogHelper.Log(LogTarget.File, $"Exception on Meetings/ScheduleMeeting endpoint method caught; error_msg: {ex.Message}");
    throw ex;
}
```

Slika 5.11: Primjer poziva zapisivača u programu

5.6 *Sastanci*

Grupacija klasa pod nazivom sastanci *(eng. Meetings)* jedna je od glavnih u ovom radu, a svodi se na nekolicinu klasa vidljivih na slici 5.12. Stvorene su s namjenom da definiraju tipove varijabli i njihove nazive kako bi se mogli pravilno popunjavali JSON objekti te kako bi na jednom mjestu postojala kolekcija svih metoda koje pozivaju API za odjeljak sastanci. Iznad svake varijable, kako bi se dodijelilo njen JSON naziv, dodan je *JsonProperty* na način koji je vidljiv na slici 5.13.



Slika 5.12: Grupacija klasa pod imenom Meetings

5.6.1 *Objekt sastanka*

Klasa objekt sastanka ili *MeetingObject* sadrži parametre koji se prikazuju prilikom dohvata sastanaka, kao što su: broj stranica, broj jedne stranice, veličina stranice, ukupni zapisi, žeton za drugu stranicu i lista objekata koju stvaramo od klase *Meeting* kao što je vidljivo sa slici 5.13.

```
11 references
public class MeetingObject
{
    [JsonProperty(PropertyName = "page_count", Order = 1)]
    2 references
    public int? PageCount { get; set; }

    [JsonProperty(PropertyName = "page_number", Order = 2)]
    2 references
    public int? PageNumber { get; set; }

    [JsonProperty(PropertyName = "page_size", Order = 3)]
    3 references
    public int? PageSize { get; set; }

    [JsonProperty(PropertyName = "total_records", Order = 4)]
    1 reference
    public int? TotalRecords { get; set; }

    [JsonProperty(PropertyName = "next_page_token", Order = 5)]
    3 references
    public string  NextPageToken { get; set; }

    [JsonProperty(PropertyName = "meetings", Order = 6)]
    public List<Meeting> Meetings = null;
}
```

Slika 5.13: Izvorni kod klase MeetingObject

5.6.2 ***Sastanak***

Klasa sastanak *(eng. Meeting)* sadrži sve parametre koji su nam potrebni kako bi se jedan sastanak prikazao, neki od parametara su obavezni, dok drugi ako su prazni neće biti popunjeni i njihova vrijednost bude iznosila *null* te će se zanemarivati.

5.6.3 ***Zahtjevno tijelo sastanka***

Klasa zahtjevno tijelo sastanka ili *MeetRequestBody* sadrži postavke koje korisnik može promijeniti prilikom ažuriranja sastanka kao što su: datum početka, lozinka, drugi voditelj (ako imate plaćenu verziju) itd.

5.6.4 ***Postavke***

Klasa postavke sastanka ili *MeetSettings* sadrži kolekciju proizvoljnih parametara koji su potrebni prilikom kreacije sastanka ili ažuriranja i to samo ako ih korisnik želi ispuniti.

5.6.5 ***Pojava***

Klasa pojava *(eng. Occurrence)* sadrži postavke koje korisnik koristi ako postoji meeting koji se već ponavlja kako bi saznao njegov status.

5.6.6 *Ponavljanje*

Klasa ponavljanje *(eng. Recurrence)* sadrži postavke koje korisnik može promijeniti ukoliko želi od sastanka koji već postoji ili od sastanka koji se tek kreira da se ponovi ili da se uzastopno ponavlja u nekom većem intervalu, što je korisno za tvrtke.

5.6.7 *Klasa sastanci*

U klasi sastanci nalaze se metode obrađene u ovom radu koje služe za: ispis svih postojećih sastanaka, dohvat samo jednog sastanka, brisanje sastanaka, kreiranje sastanaka i ažuriranje sastanka. Na slikama 5.14 i 5.15, vidljive su metode: *ScheduleMeeting* u kojoj preko *Post* zahtjeva pokušavamo napraviti sastanak i metoda *GetMeeting* koja preko *Get* poziva dohvaća informacije za jedan sastanak.

```
1 reference
public string ScheduleMeeting(MeetRequestBody inputObj)
{
    string retVal = "";
    APIHandler apiHandler = new APIHandler();

    string bodyRequest = JsonConvert.SerializeObject(inputObj);
    var bodyReqContent = new StringContent(bodyRequest, Encoding.UTF8, "application/json");

    try
    {
        retVal = apiHandler.Post($"{Meetings.userUrlAddOn}/{APIHandler.userId}/{Meetings.mettingsEndpoint}", bodyReqContent);
    }
    catch (Exception ex)
    {
        Console.WriteLine(FileLogger.filePath);
        LogHelper.Log(LogTarget.File, $"Exception on Meetings/ScheduleMeeting endpoint method caught; error_msg: {ex.Message}");
        throw ex;
    }
    return retVal;
}
```

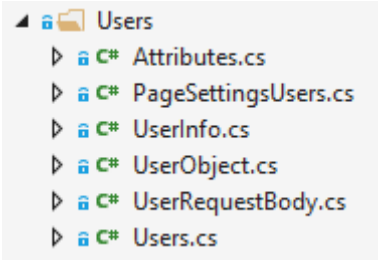
Slika 5.14: Izvorni kod metode ScheduleMeeting

```
3 references
public Meeting GetMeeting(Int64? meetingId)
{
    APIHandler apiHandler = new APIHandler();
    Meeting retVal = null;

    try
    {
        string response = apiHandler.Get($"{Meetings.mettingsEndpoint}/{meetingId}");

        retVal = JsonConvert.DeserializeObject<Meeting>(response);

    }
    catch (Exception ex)
    {
        Console.WriteLine(FileLogger.filePath);
        LogHelper.Log(LogTarget.File, $"Exception on Meetings/GetMeeting endpoint method caught; error_msg: {ex.Message}");
        throw ex;
    }
    return retVal;
}
```

Slika 5.15: Izvorni kod metode GetMeeting

5.7 *Korisnici*

Grupacija klasa pod imenom korisnici *(eng. Users)* je druga glavna grupacija klasa u ovom radu. Sastoji se od šest klasa koje su vidljive na slici 5.16, a njihova namjena je definiranje tipova varijabli vezanih uz korisnika za popunjavanje JSON objekata, kao i spremište za metode koje nam služe za npr. dohvaćanje jednog korisnika ili više njih, ažuriranje itd.



Slika 5.16: Grupacija klasa pod imenom Users

5.7.1 ***Postavke stranice korisnika***

Klasa postavke stranice korisnika *(eng. user page settings)* ili *PageSettingsUser* koristi nam slično kao što je navedeno u 5.6.1, samo što su to ovaj put informacije o korisniku i lista stvorena od klase *UserObjec*t o kojoj se govori u narednom potpoglavlju.

5.7.2 ***Objekt korisnika***

Klasa objekt korisnika *(eng. user object)* ili *UserObject* sastoji se od parametara koji su popunjeni prilikom dohvata korisnika s API-a. Slično kao i kod sastanaka, dodani su „?“ poslije tipova parametara koji su brojčanog tipa, kako bi mogli biti *null* vrijednosti i kako bi ih se na taj način u programu moglo zanemariti.

5.7.3 ***Informacije korisnika***

Klasa informacije korisnika *(eng. user info)* ili *UserInfo* sadrži osnovne parametre o nekom korisniku te nam služi da prilikom kreacije korisnika popunimo njegovo ime i prezime.

5.7.4 ***Klasa korisnici***

Klasa korisnici *(eng. Users)* sadrži kolekciju metoda potrebnih kako bi: ispisali sve korisnike, napravili korisnika, dohvatili korisnika po elektroničkoj adresi, obrisali korisnika i ažurirali korisnika. Na slikama 5.17, 5.18 i 5.19 vidljive su metode za dohvat

jednog korisnika, brisanje korisnika i ažuriranje korisnika o kojima govorimo u narednim potpoglavljima.

5.7.5 *Dohvati korisnika Zoom računa*

Kao što vidimo na slici 5.17, *GetUser* objekt prima tekstualnu varijablu *userId* te „pokušava“ obaviti „poziv“. *JsonConvert.DeserializeObject* koristimo kako bismo JSON podatke dobili u pravom formatu.

```
2 references
public UserObject GetUser(string userId)
{
    UserObject retVal = null;
    try
    {
        APIHandler apiHandler = new APIHandler();
        string response = apiHandler.Get($"{Users.userUrlAddOn}/{userId}");
        retVal = JsonConvert.DeserializeObject<UserObject>(response);

    }
    catch (Exception ex)
    {
        LogHelper.Log(LogTarget.File, $"Exception on Users/GetUser; error_msg:  {ex.Message}");
        throw ex;
    }
    return retVal;
}
```

Slika 5.17: Izvorni kod metode GetUser

5.7.6 *Obriši korisnika Zoom računa*

Obriši korisnika funkcionira po sličnom principu kao što je opisano u 5.7.5 i kao što je vidljivo na slici 5.18. Jedina razlika između *GetUser* i *DeleteUser* je u tome što je *GetUser* definiran kao objekt te se koristi kako bi se dohvatili podaci o korisniku, za razliku od *DeleteUser* metode koja se koristi samo kako bi se korisnik obrisao, za što nam nije potreban objekt.

```
1 reference
public string DeleteUser(string _userId)
{
    string retVal = "";
    try
    {
        APIHandler apiHandler = new APIHandler();
        retVal = apiHandler.Delete($"{Users.userUrlAddOn}/{_userId}");
    }
    catch (Exception ex)
    {
        LogHelper.Log(LogTarget.File, $"Exception on Users/DeleteUser; error_msg:  {ex.Message}");
        throw ex;
    }
    return retVal;
}
```

Slika 5.18: Izvorni kod metode DeleteUser

5.7.7 *Ažuriraj korisnika Zoom računa*

Ažuriraj korisnika Zoom računa ili metoda *UpdateUser* prima tekstualni identifikator *_userId* te objekt *userUpdateObject*. Svrha ove metode je ažurirati podatke o korisniku kojeg smo dodali na naš Zoom račun. Prvo moramo serijalizirati objekt *(eng. serialize object)* kako bi objekt poslali na Zoom API u pravilnom formatu, nakon toga dolazi *queryParameters* parametar koji se stvara unutar same metode i on je lista ključeva *(eng. key)* potrebnih kako bi definirali tip prijave i samim time dobili pravo na ažuriranje korisnika. U zahtjevu opisanom u potpoglavlju 5.3.4, prosljeđujemo krajnju točku koja se sastoji od *userUrlAddOn* varijable,*_userId* varijable te samog objekta i *query* parametara kao što je vidljivo na slici 5.19.

```
2 references
public string UpdateUser(string _userId, UserObject userUpdateObject)
{
    string retVal = "";
    try
    {
        APIHandler apiHandler = new APIHandler();

        string bodyRequest = JsonConvert.SerializeObject(
            userUpdateObject,
            Newtonsoft.Json.Formatting.None,
            new JsonSerializerSettings
            {
                NullValueHandling = NullValueHandling.Ignore
            }
        );

         var bodyReqContent = new StringContent(bodyRequest, Encoding.UTF8, "application/json");

        List<KeyValuePair<string, string>> queryParameters = new List<KeyValuePair<string, string>>()
        {
            new KeyValuePair<string, string>("login_type", "1")
        };

        retVal = apiHandler.Patch($"{Meetings.userUrlAddOn}/{_userId}", bodyReqContent, queryParameters);
    }
    catch (Exception ex)
    {
        LogHelper.Log(LogTarget.File, $"Exception on Users/UpdateUser; error_msg:  {ex.Message}");
        throw ex;
    }
    return retVal;
}
```

Slika 5.19: Izvorni kod metode UpdateUser

5.8 *Testiranje*

Kako i za većinu projekata tako i za ovaj vrijedi činjenica da mora proći fazu ispitivanja pogrešaka na njemu kako bi se mogao dalje koristiti. U ovom slučaju smo za testiranje koristili konzolnu aplikaciju.

5.8.1 *Počeci*

Prije bilo kakvog testiranja na aplikaciji morali smo fiksno definirati glavne parametre koji su vidljivi na slici 5.20 i koji se ne mijenjaju ali nam trebaju prilikom svakog poziva API-a. Ti parametri su: *userId, zoomApiBaseUrl, zoomApiUrl i jwtToken*. Pošto u trenutku testiranja prvih metoda za dohvat sastanaka i korisnika nije još postojalo grafičko korisničko sučelje niti prijava, parametri su morali biti fiksno zadani kako bi se metode mogle izvršavati.

```
APIHandler.userId = "dkovacevic@vub.hr";
APIHandler.zoomApiBaseUrl = "https://api.zoom.us/v2";
APIHandler.zoomApiUrl = $"{APIHandler.zoomApiBaseUrl}";
APIHandler.jwtToken = "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJhdWQiOm51bGwsImlzcyI6
```

Slika 5.20: Fiksni parametri za testiranje aplikacije

5.8.2 *Proces*

Kako bismo testirali našu aplikaciju, stvaramo objekte unutar glavnog programa koji se šalju u metodama, npr. ako želimo promijeniti ime i prezime korisnika ili samo titulu njegovog posla, moramo napraviti objekt koji sadrži te informacije u sebi te ga proslijediti metodi koja prima takav tip objekta i vraća nam uspješan ili neuspješan poziv API-a kao što je vidljivo na slici 5.21.

```
UserObject userUpdateObject = new UserObject()
{
    FirstName = "Ime",
    JobTitle = "Opis posla",
};

users.UpdateUser("domixzoR123@gmail.com", userUpdateObject);
```

Slika 5.21: Testiranje metode UpdateUser

5.9 *Konfiguracija*

Konfiguracija *(eng. configuration)* je grupacija klasa korištenih za brzu prijavu. Naime, prilikom svake prijave korisnika, ukoliko je uspješna, stvara se JSON datoteka u kojoj su upisani podaci korisnika, konfiguracija funkcionira po sličnom principu kao što je opisano u 5.5.

5.9.1 *Baza konfiguracije*

Baza konfiguracije *(eng. configuration base)* ili klasa *ConfigBase* služi nam za definiranje putanje na kojoj će se stvoriti JSON datoteka i za dio programskog koda

potreban kako bi se u tu datoteku upisivalo te iz nje čitalo kao što je vidljivo na slici 5.22. Klasa sadrži apstraktne metode *WriteConfig* i *ReadConfig*. *WriteConfig* prima poruku u tekstualnom obliku koja će se upisati u datoteku, dok *ReadConfig* metoda služi samo za čitanje iz datoteke i ne prima parametre. Kao i kod *LogBase* klase opisane u potpoglavlju 5.5.2, klasa nam koristi kako bi odredili tip podatka kojeg kreiramo, u ovom slučaju je to *.json*. Konfiguracijska datoteka stvara se u direktoriju u kojem se pokreće aplikacija.

```
2 references
public abstract class ConfigBase
{
    protected readonly object lockObj = new object();
    2 references
    public abstract void WriteConfig(string message);

    2 references
    public abstract string ReadConfig();
}

2 references
public class ConfigHandler : ConfigBase
{
    private static string fileName = "config";

    private DirectoryInfo AddOnDirectory = Directory.CreateDirectory($"{AppDomain.CurrentDomain.BaseDirectory}Config");

    public static string filePath = $"{AppDomain.CurrentDomain.BaseDirectory}Config\\{fileName}.json";
    2 references
    public override void WriteConfig(string message)
    {
        lock (lockObj)
        {
            using (StreamWriter streamWriter = new StreamWriter(filePath))
            {
                streamWriter.WriteLine(message);
                streamWriter.Close();
            }
        }
    }

    2 references
    public override string ReadConfig()
    {
        lock (lockObj)
        {
            using (StreamReader streamReader = new StreamReader(filePath))
            {
                string strJson = streamReader.ReadToEnd();
                return strJson;
            }
        }
    }
}
```

Slika 5.22: Izvorni kod klase ConfigBase

5.9.2 ***Pomoćnik konfiguracije***

Pomoćnik konfiguracije *(eng. configuration helper)* ili *ConfigHelper* je klasa koja radi po principu opisanom u potpoglavlju 5.5.4 i kao što vidimo na slici 5.23.

```
2 references
public static class ConfigHelper
{
    private static ConfigBase configBase = null;
    1 reference
    public static void WriteConfig(string message)
    {
        configBase = new ConfigHandler();
        configBase.WriteConfig(message);
    }

    1 reference
    public static string ReadConfig()
    {
        configBase = new ConfigHandler();
        return configBase.ReadConfig();
    }
}
```

Slika 5.23: Izvorni kod klase ConfigHelper

5.9.3 ***Objekt konfiguracije***

Objekt konfiguracije *(eng configuration object)* ili *ConfigObject* klasa koristi nam za pohranjivanje JSON varijabli kao što je vidljivo na slici 5.24.

```
4 references
class ConfigObject
{
    [JsonProperty(PropertyName = "user_id")]
    2 references
    public string UserId { get; set; }
    [JsonProperty(PropertyName = "api_key")]
    2 references
    public string ApiKey { get; set; }
    [JsonProperty(PropertyName = "api_secret")]
    2 references
    public string ApiSecret { get; set; }
    [JsonProperty(PropertyName = "expiration_date")]
    1 reference
    public long ExpDate { get; set; }

}
```

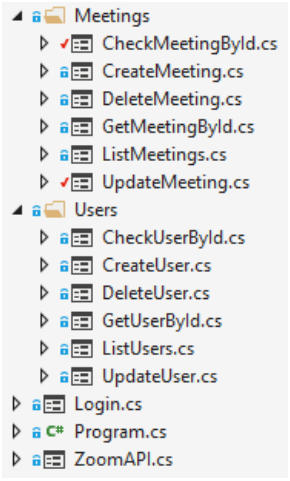
Slika 5.24: Izvorni kod klase ConfigObject

5.10 ***Grafičko korisničko sučelje***

Grafičko korisničko sučelje *(eng. Graphical User Interface)* korišteno u ovom radu spomenuto je u poglavlju 2.3 i u narednim potpoglavljima će biti opisano na koji način funkcioniraju obrasci u našem projektu.

5.10.1 *Općenito*

Korištenjem GUI-a doprinosi se korisničkom iskustvu. U svrhu izrade ovog rada kreirano je četrnaest obrazaca s kojima korisnik može doći u kontakt. Kao što možemo vidjeti na slici 5.25, opet su kreirane grupacije posebno za sastanke i posebno za korisnike te klase *Login* i *ZoomAPI*.



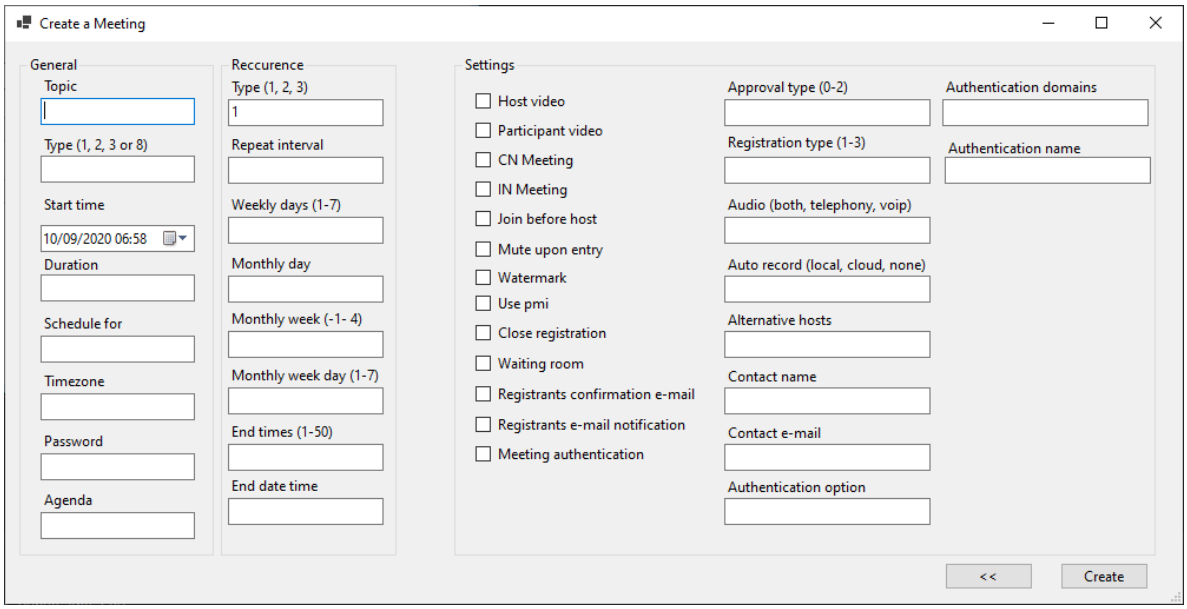
Slika 5.25: Obrasci

5.10.2 *Provjeri sastanak po identifikatoru*

Provjeri sastanak po identifikatoru ili *CheckMeetingById* je obrazac koji se koristi za dohvaćanje podataka o sastanku. U trenutku kad je poslan *meetingId*, popunjava se obrazac *UpdateMeeting* kako bi korisnik mogao samo promijeniti informacije u usporedbi s pisanjem novih.

5.10.3 *Kreiraj sastanak*

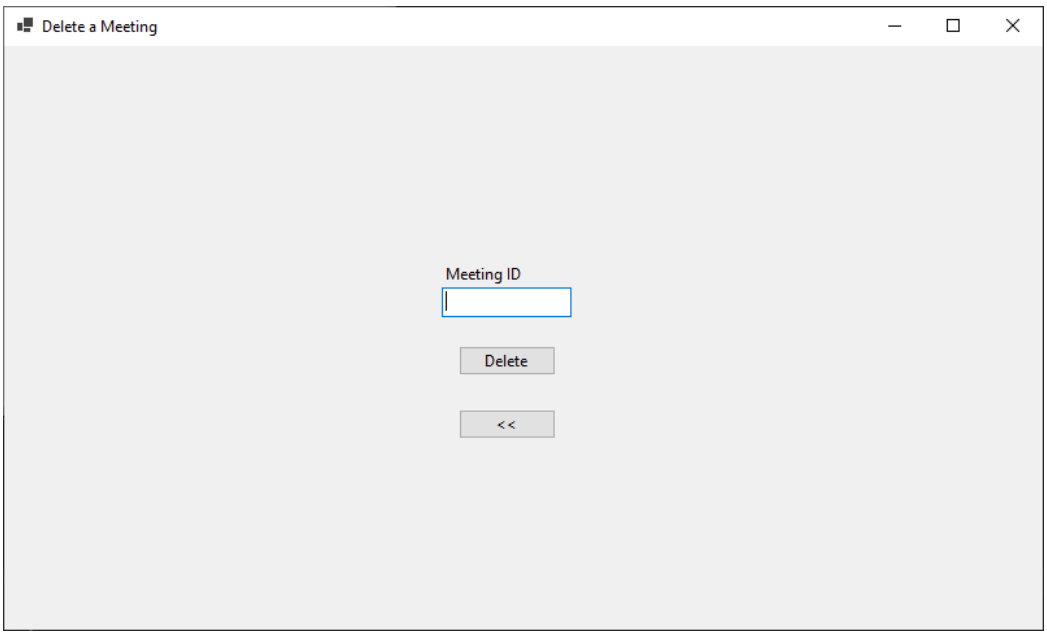
Kreiraj sastanak ili *CreateMeeting* je obrazac koji korisnik popunjava ako želi stvoriti sastanak. Sve što korisnik treba učiniti je ispuniti polje *Topic* i pritisnuti *Create* tipku kao što je vidljivo na slici 5.26.



Slika 5.26: Create a Meeting obrazac

5.10.4 ***Obriši sastanak***

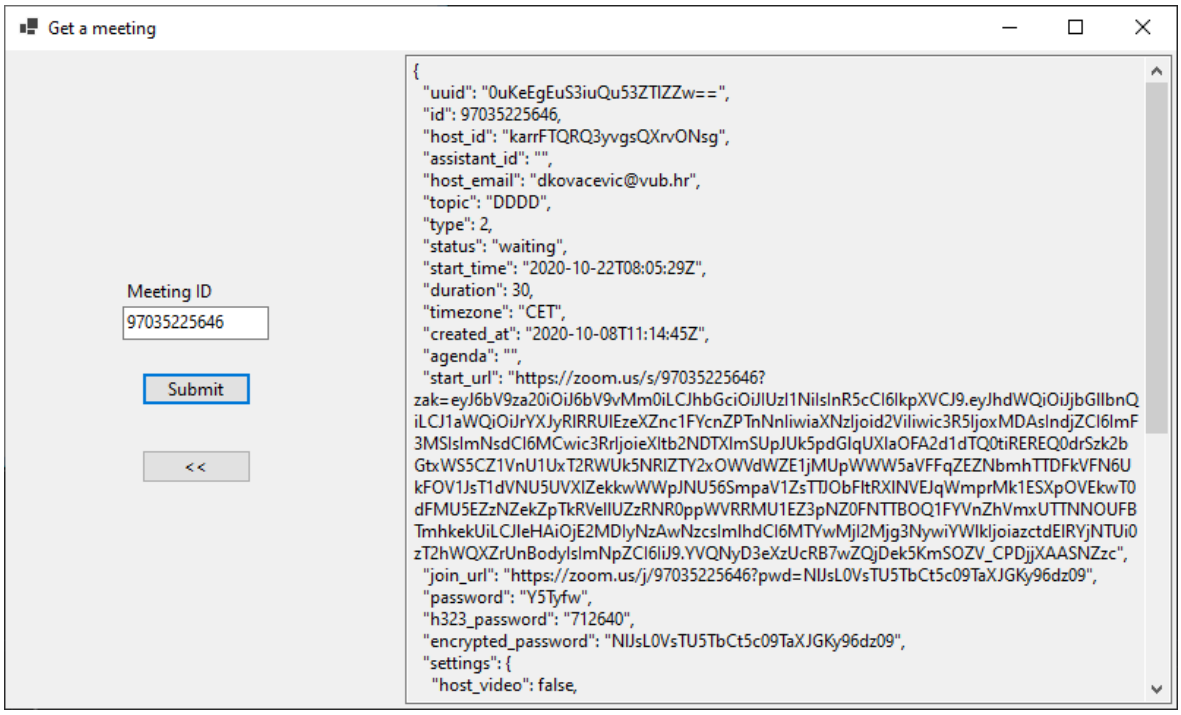
Obriši sastanak ili *DeleteMeeting* je obrazac s jednim poljem u koje korisnik unosi *Meeting ID* te ispod na pritisak prve tipke brišemo sastanak ako postoji, a na pritisak druge vraćamo se natrag kao što je vidljivo na slici 5.27.



Slika 5.27: Delete a Meeting obrazac

5.10.5 *Uhvati sastanak po identifikatoru*

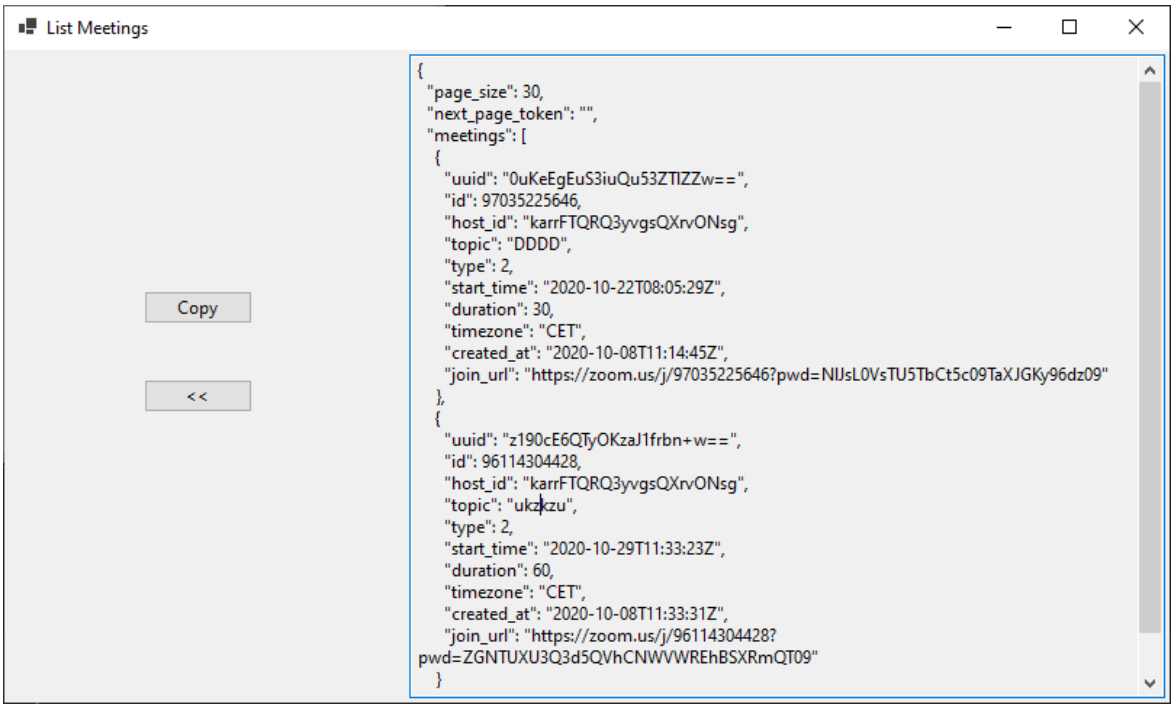
Uhvati sastanak po identifikatoru ili *GetMeetingById* obrazac služi nam kako bi dohvatili podatke o jednom sastanku u JSON formatu kao što vidimo na slici 5.28.



Slika 5.28: Get a meeting obrazac

5.10.6 *Pokaži sastanke*

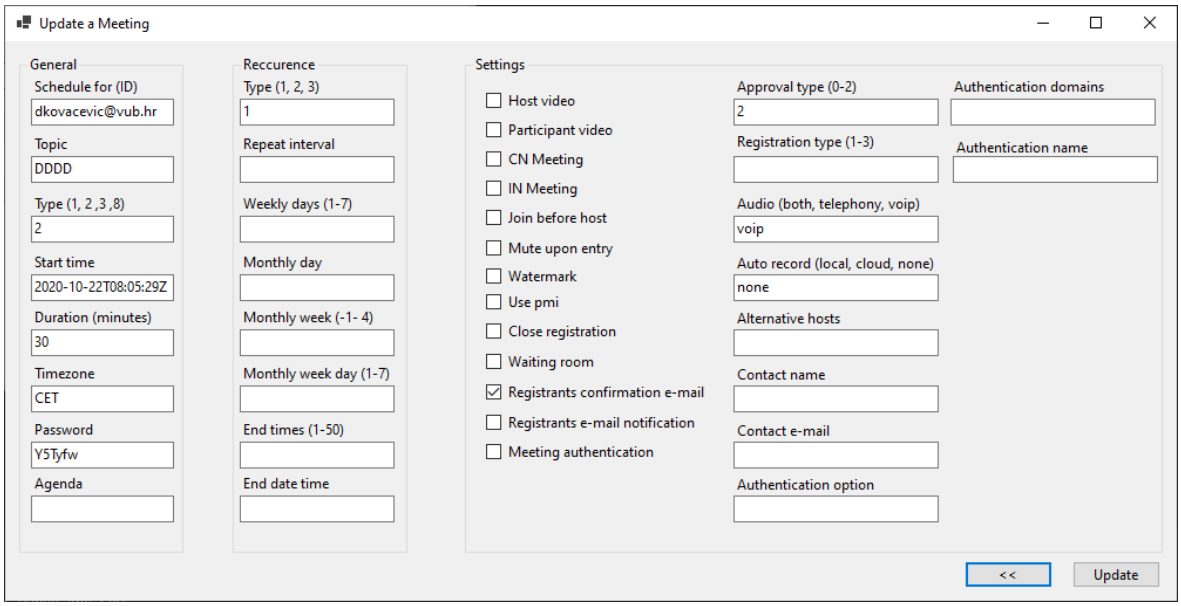
Pokaži sastanke ili *ListMeetings* obrazac koristi nam kao ekran na kojem možemo vidjeti sve postojeće sastanke u JSON formatu te ih odmah kopirati putem pritiska na gumb *Copy* kao što je vidljivo na slici 5.29.



Slika 5.29: List Meetings obrazac

5.10.7 ***Ažuriraj sastanak***

Ažuriraj sastanak ili *UpdateMeeting* je obrazac na kojem postoje različita polja za unos podataka kako bi mogli promijeniti npr. vrijeme početka sastanka. Kao što vidimo iz slike 5.30.



Slika 5.30: Update a Meeting obrazac

5.10.8 *Provjeri korisnika po identifikatoru*

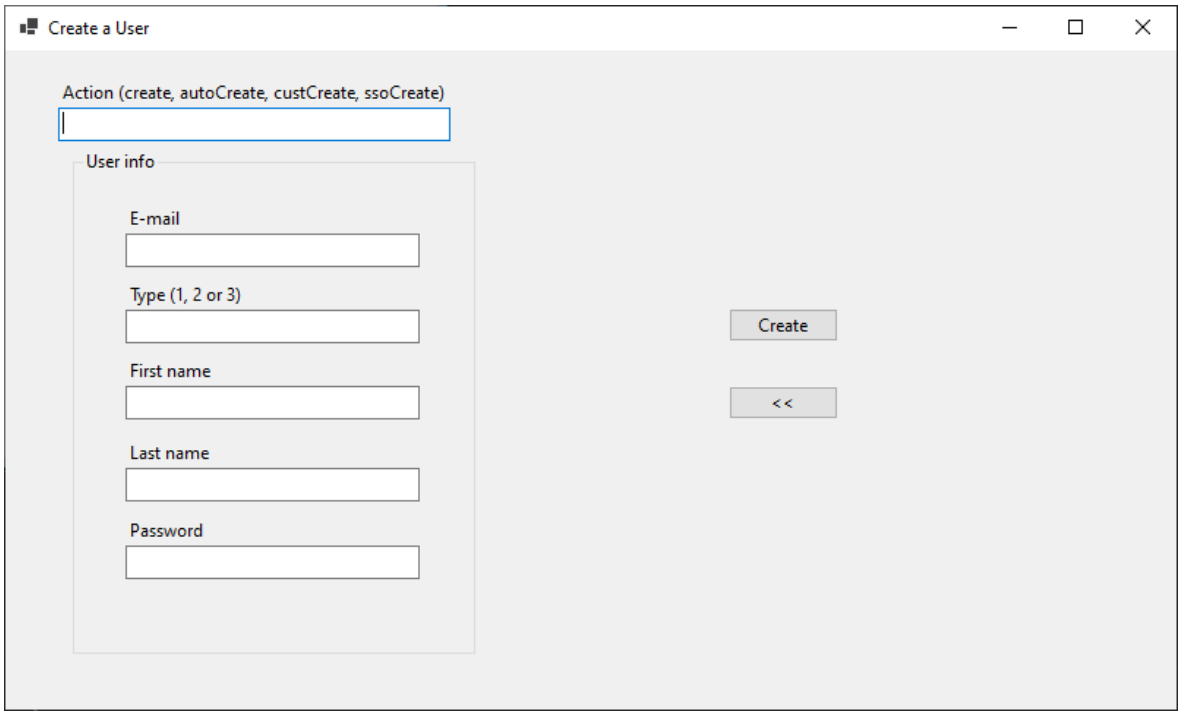
Provjeri korisnika po identifikatoru ili *CheckUserById* je obrazac koji radi po sličnom principu kao i *CheckMeetingById* je opisan u potpoglavlju 5.9.2. Unosom ispravne e-mail adrese, korisnik može provjeriti drugog korisnika po identifikatoru te ga aplikacija preusmjeri na obrazac za npr. ažuriranje.

5.10.9 *Obriši korisnika*

Obriši korisnika ili *DeleteUser* je obrazac koji nam služi za brisanje korisnika koje smo kreirali preko aplikacije. Funkcionira po sličnom principu kao i *DeleteMeeting* koji je opisan u potpoglavlju 5.9.4.

5.10.10 *Kreiraj korisnika*

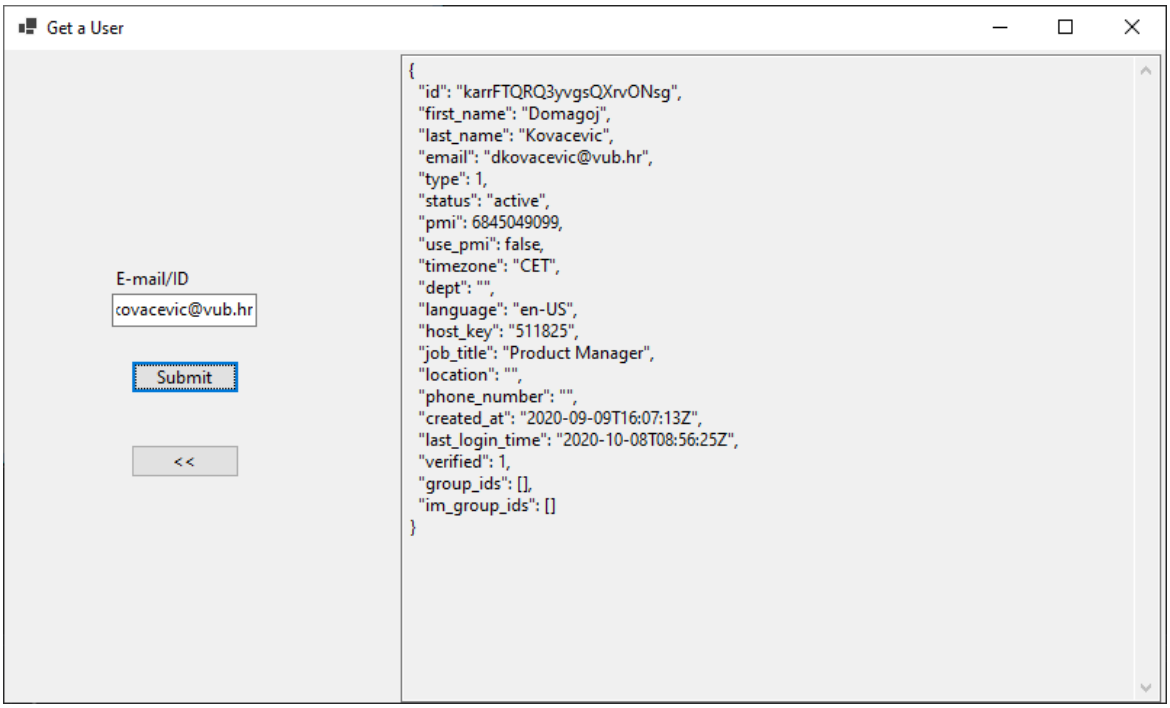
Kreiraj korisnika ili *CreateUser* obrazac nam služi kako bismo korisniku kojeg želimo dodati poslali Zoom e-mail u kojem mu se referenciramo imenom i prezimenom koje smo popunili na obrascu kao što možemo vidjeti na slici 5.31.



Slika 5.31: Create a User obrazac

5.10.11 *Uhvati korisnika po identifikatoru*

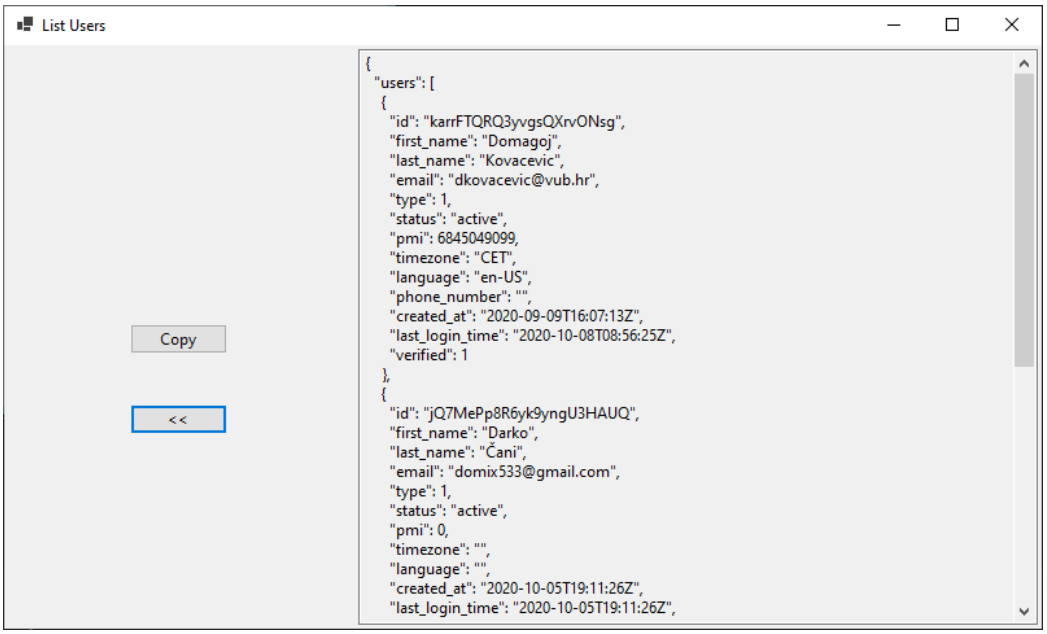
Uhvati korisnika po identifikatoru ili *GetUserById* je obrazac koji služi za dohvaćanje informacija o jednom korisniku u JSON formatu. Pritiskom na tipku *Copy* možemo prekopirati njegove informacije kao što je vidljivo na slici 5.32.



Slika 5.32: Get a User obrazac

5.10.12 ***Pokaži korisnike***

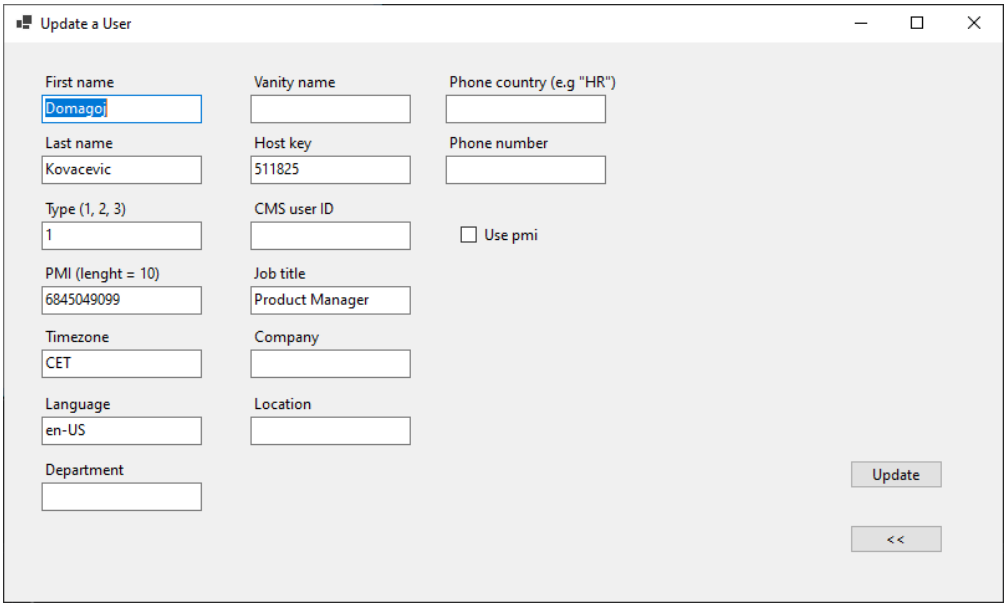
Pokaži korisnike ili *ListUsers* je obrazac koji prikazuje sve korisnike u JSON formatu te ih također lako možemo prekopirati kako je navedeno u 5.10.11 i kao što je vidljivo na slici 5.33.



Slika 5.33: List Users obrazac

5.10.13 *Ažuriraj korisnika*

Ažuriraj korisnika ili *UpadeUser* je obrazac na kojemu možemo mijenjati informacije o korisniku kao što vidimo na slici 5.34. Pritiskom na *Update* dobivamo obavijest koja nam govori da je korisnik uspješno ažuriran.



Slika 5.34: Update a User obrazac

6. ZAKLJUČAK

Ovaj završni rad predstavlja razvoj biblioteke za Zoom API. Temelji se na kolegiju C# programiranje. Ciljevi završnog rada su sljedeći: stvaranje interaktivne C# biblioteke koja se može koristiti u svrhu razvijanja programa koji pozivaju Zoom-ov API. Dodatno je kreirano grafičko korisničko sučelje za bolje korisničko iskustvo i prezentiranje biblioteke. Pri izradi biblioteke korišteni su: programsko okruženje Visual Studio 2019, programski jezik C# i *Windows Forms*. Kreacijom nekolicine metoda koje služe za povezivanje sa Zoom API-em te kreacijom klasa i potklasa za pravilno prihvaćanje i ispis JSON varijabli stvorena je biblioteka uz čiju pomoć možemo dohvatiti bilo koju krajnju točku za: kreiranje korisnika i sastanaka, ažuriranje korisnika i sastanaka, isto kao i dohvaćanje i brisanje istih. Završni rad predstavlja prototip aplikacije koju može koristiti Veleučilište u Bjelovaru ili bilo koja druga institucija kako bi na efektivniji način kreirali, ažurirali te dohvatili sastanke i korisnike na Zoom računu. S obzirom na današnju epidemiološku situaciju te izvođenje nastave na daljinu, Biblioteka za Zoom API pridonosi jednostavnosti izvođena iste i olakšava način za organizaciju sastanaka i korisnika Zoom platforme unutar institucije.

7. LITERATURA

[1] Development environment [Online].2020.

Dostupno na:

https://searchsoftwarequality.techtarget.com/definition/development-environment

[2] What is Visual Studio? [Online].2020.

Dostupno na: https://www.cbronline.com/what-is/what-is-visual-studio-4959054/

[3] Chad M.: What is C#[Online].2020.

Dostupno na: https://www.c-sharpcorner.com/article/what-is-c-sharp/

[4] Introduction to C# Windows Forms Applications [Online].2019.

Dostupno na:

https://www.geeksforgeeks.org/introduction-to-c-sharp-windows-forms-applications/

[5] Widnows Forms overview [Online].2017.

Dostupno na:

https://docs.microsoft.com/en-us/dotnet/desktop/winforms/windows-forms-overview?view=netframeworkdesktop-4.8

[6] Zoom explained: Understanding (and using) the popular video chat app [Online].2020.

Dostupno na:

https://www.computerworld.com/article/3570623/the-zoom-meeting-app-explained-understanding-and-using-the-popular-video-chat-software.html

[7] What is Zoom and how does it work? Plus tips and tricks [Online].2020.

Dostupno na:

https://www.pocket-lint.com/apps/news/151426-what-is-zoom-and-how-does-it-work-plus-tips-and-tricks

[8] Zoom API [Online].2020.

Dostupno na: https://marketplace.zoom.us/docs/api-reference/zoom-api

[9] Zoom App Marketplace [Online].2020.

Dostupno na: https://marketplace.zoom.us/

[10] JWT [Online].2020.

Dostupno na: https://jwt.io/

[11] What is an API? [Online].2020.

Dostupno na:

https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces

[12] Freeman J.: What is an API? Application programming interfaces explained [Online].2019.
Dostupno na:
https://www.infoworld.com/article/3269878/what-is-an-api-application-programming-interfaces-explained.html
[13] Devero A.: Programming languages, libraries and frameworks [Online].2015.
Dostupno na:
https://blog.alexdevero.com/programming-languages-libraries-and-frameworks/#:~:text=%E2%80%9C.%E2%80%9D)%20%7D-,What%20is%20a%20library,classes%2C%20values%20or%20type%20specifications.
[14] Wexelblat R.: History of Programming Languages [Online].1981.
Dostupno na: https://archive.org/details/historyofprogram0000hist/page/n791/mode/2up
[15] What does „library“ mean in case of programming languages? [Online].2016.
Dostupno na:
https://www.quora.com/What-does-library-mean-in-the-case-of-programming-languages
[16] What is the Difference Between Web Services and API? [Online].2019.
Dostupno na:
https://nordicapis.com/what-is-the-difference-between-web-services-and-apis/

8. OZNAKE I KRATICE

API – *eng. Application Programming Interface*

CSS – *eng. Cascading Style Sheets*

HTML – *eng. HyperText Markup Language*

HTTPS – *eng. HyperText Transfer Protocol Secure*

itd.– i tako dalje

JSON – *eng. JavaScript Object Notation*

JWT – *eng. JSON Web Token*

npr. – na primjer

i sl. – i slično

REST – *eng. Representational state transfer*

UI – *eng. User Interface*

UDDI – *eng. Universal Description, Discovery and Integration*

SOAP – *eng. Simple Object Access Protocol*

XML-RPC – *eng. XML Remote Procedure Call*

9. SAŽETAK

Naslov: Biblioteka za Zoom API

U ovom završnom radu prikazujemo izradu biblioteke za Zoom API, te pripadajućeg grafičkog korisničkog sučelja kako bi se metode koje biblioteka sadrži mogle prikazati vizualno. Ona se sastoji od nekolicine metoda te klasa i potklasa napisanih u programskom jeziku C# koje služe kao omotač *(eng. wrapper)*. Ideja je stvoriti metode od kojih samo neke služe za kreiranje i brisanje sastanaka, te pozivanje na isti. Zoom API dozvoljava nam pristup informacijama platforme Zoom. Cilj biblioteke je pojednostavljenje održavanja nastave na daljinu na Veleučilištu u Bjelovaru. Rezultat je aplikacija kojom možemo efektivno organizirati sastanke i ažurirati iste, te isto tako i korisnike.

Ključne riječi: Zoom, API, biblioteka, C#, JWT, JSON, žeton, zapisivač.

10. ABSTRACT

Title: Zoom API library

In this final paper, we present the creation of a library for the Zoom API, and the associated graphical user interface so that the methods contained in the library can be displayed visually. It consists of several methods, classes and subclasses written in the C# programming language that serve as a wrapper. The idea is to create methods, only some of which are used to create and delete meetings, and to refer to them. The Zoom API allows us to access Zoom platform information. The aim of the library is to simplify remote learning at the Bjelovar University of Applied Sciences. The result is an application with which we can effectively organize meetings and update them, as well as users.

Keywords: Zoom, API, library, C#, JWT, JSON, token, logger.

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 19. 10. 2020.	DOMAGOJ KOVAČEVIĆ	Kovačević

Prema Odluci Veleučilišta u Bjelovaru, a u skladu sa Zakonom o znanstvenoj djelatnosti i visokom obrazovanju, elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru bit će pohranjene i javno dostupne u internetskoj bazi Nacionalne i sveučilišne knjižnice u Zagrebu. Ukoliko ste suglasni da tekst Vašeg završnog rada u cijelosti bude javno objavljen, molimo Vas da to potvrdite potpisom.

Suglasnost za objavljivanje elektroničke inačice završnog rada u javno dostupnom nacionalnom repozitoriju

DOMAGOJ KOVAČEVIĆ

ime i prezime studenta/ice

Dajem suglasnost da se radi promicanja otvorenog i slobodnog pristupa znanju i informacijama cjeloviti tekst mojeg završnog rada pohrani u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu i time učini javno dostupnim.

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 19.10.2020.

potpis studenta/ice