SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Leo Ćavar

IZRADA APLIKACIJE ZA PRONALAZAK TERMINA SASTANAKA

ZAVRŠNI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Leo Ćavar

Matični broj: 0016153823

Studij: Informacijski i poslovni sustavi

IZRADA APLIKACIJE ZA PRONALAZAK TERMINA SASTANAKA

ZAVRŠNI RAD

Mentor :

Doc. Dr. sc. Marko Mijač

Varaždin, Rujan 2024.

Leo Ćavar

Izjava o izvornosti

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

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U ovom završnom radu obrađuje se korištenje .NET tehnologija za izradu programa za dogovaranje sastanaka. Rad pokriva kratki pregled ASP.NET Core i ASP.NET Web API tehnologija, uz objašnjenje koncepata API-ja i HTTP metoda. Poseban naglasak stavljen je na korištenje Google API-ja, uključujući Google Calendar API, te autentifikaciju i autorizaciju korisnika pomoću OAuth 2.0 protokola. Kroz rad se također prikazuju primjeri zahtjeva za dohvaćanje i upravljanje kalendarskim događajima, uz detaljnu implementaciju .NET web aplikacije koja omogućuje dogovaranje sastanaka među korisnicima.

Ključne riječi: API; ASP. NET Core; .NET; C#; ASP. NET; ; OAuth 2.0; Google Calendar; Google API;

Sadržaj

1. Uvod

U ovom radu usredotočit ćemo se na izradu programa za dogovaranje sastanaka putem Google API-ja, koji omogućuje jednostavnu interakciju s iznimno popularnim Google kalendarom. Google kalendar nudi niz funkcionalnosti koje olakšavaju organizaciju i planiranje događaja, cilj ovoga rada je iskoristiti te funkcionalnosti kako bismo omogućili lakše usklađivanje termina između više sudionika. Kroz implementaciju ovog programa, korisnici će moći u stvarnom vremenu pregledavati slobodne termine svih sudionika te na temelju tih podataka dogovarati sastanke na način koji svima odgovara. Program ćemo izraditi korištenjem ASP.NET Core frameworka, koji omogućuje razvoj web aplikacija. Razor stranice koristit ćemo za izradu korisničkog sučelja i upravljanje podacima, stvarajući intuitivno korisničko iskustvo. S druge strane, ASP.NET Web API služit će za primanje HTTP zahtjeva i komunikaciju s Google servisima, omogućujući manipulaciju događajima u kalendaru. Na taj način, uspostavit ćemo interakciju između korisnika i Google API-ja, automatizirajući procese koji bi se inače obavljali ručno. Poseban naglasak ovog rada bit će na integraciji Google Calendar API-ja s našom aplikacijom. Ovaj API omogućuje pristup informacijama o zauzetim i slobodnim terminima, kao i kreiranje, ažuriranje i brisanje događaja. Kroz ovaj projekt prikazat ćemo tehničke aspekte integracije s Google Calendar API-jem, uključujući autentifikaciju putem Googleovog OAuth 2.0 protokola te dohvaćanje i obradu podataka o događajima u kalendaru. Ova aplikacija će omogućiti učinkovitije planiranje sastanaka, smanjujući vrijeme potrebno za pronalaženje zajedničkih slobodnih termina, te će na taj način olakšati korisnicima svakodnevne organizacijske zadatke.

2. Problem pronalaska slobodnog termina za sastanke

U današnjem poslovnom okruženju učinkovito upravljanje vremenom od ključne je važnosti za organizaciju i koordinaciju timova, sastanaka i događanja. S obzirom na sve veći broj sastanaka i obaveza, pronalazak zajedničkog slobodnog vremena za više sudionika predstavlja izazov, osobito u organizacijama s velikim brojem zaposlenika ili suradnicima na različitim lokacijama. Ovaj problem postaje još složeniji kada se u obzir uzmu raspoloživi resursi poput dvorana, tehničke opreme ili vremena dostupnosti pojedinaca.

Pronalaženje slobodnih termina uključuje nekoliko ključnih aspekata:

- **Dostupnost sudionika**: Svaki sudionik ima vlastiti raspored koji uključuje radno vrijeme, obaveze i osobno vrijeme. Usklađivanje tih rasporeda može biti zahtjevno, pogotovo kada se radi o većem broju sudionika.
- **Raspoloživost resursa**: Pored sudionika, potrebno je uzeti u obzir dostupnost resursa poput prostorija za sastanke ili tehničke opreme, koji također mogu imati ograničenu dostupnost.

Uspješno rješavanje ovog problema ključno je za povećanje produktivnosti i smanjenje vremena potrebnog za dogovaranje sastanaka. Neefikasno upravljanje vremenom može rezultirati propuštenim prilikama, nezadovoljstvom zaposlenika te povećanim troškovima rada. S obzirom na to, mnoge organizacije nastoje automatizirati ovaj proces korištenjem digitalnih alata za zakazivanje sastanaka.

2.1. Pregled postojećih rješenja

Jedno od najpoznatijih rješenja za organizaciju sastanaka je **Microsoft Outlook**, posebno za korisnike koji koriste Microsoft 365 ili Microsoft Exchange račune. Outlook omogućuje planiranje sastanaka pomoću alata poput *Scheduling Assistant* i *Room Finder*. Ovi alati pomažu u pronalasku slobodnog vremena ne samo za sudionike sastanka, već i za raspoložive resurse, kao što su dvorane za sastanke. *Scheduling Assistant* omogućuje korisnicima pregled dostupnih termina sudionika te automatski predlaže najbolje vrijeme za sastanak prema njihovim zauzetim i slobodnim terminima. Uz to, *Room Finder* olakšava pronalazak slobodnih prostorija, ovisno o postavkama unutar Microsoft 365 ili Exchange okruženja. Iako je Outlook vrlo moćno rješenje, njegova upotreba veže korisnike na Microsoftov ekosustav, što znači da organizacije koje ne koriste Microsoft 365 ili Exchange ne mogu u potpunosti iskoristiti sve funkcionalnosti ovog alata. Osim toga, velik dio tržišta koristi *Google Calendar*, koji je mnogo pristupačniji i popularniji među korisnicima, osobito u manjim organizacijama i među individualnim korisnicima. Stoga, iako Outlook nudi napredne funkcionalnosti, njegov utjecaj je ograničen na specifičnu skupinu korisnika, dok je *Google Calendar* prisutniji na globalnom tržištu.

3. Tehnologije za razvoj rješenja

3.1. Google Cloud projekt

Kako bi se koristili *Google Workspace API*-ji i razvijale aplikacije koje koriste Google servise, potrebno je postaviti projekt unutar *Google Cloud Console*-a. Google Cloud projekt pruža osnovu za kreiranje, omogućavanje i korištenje svih Google Cloud servisa, uključujući upravljanje API-ima, omogućavanje naplate, dodavanje suradnika i upravljanje dopuštenjima. Kreiranje projekta unutar Google Cloud-a ključan je korak u integraciji aplikacija s Google Workspace servisima, jer omogućuje autentifikaciju i autorizaciju korisnika te sigurno upravljanje podacima [1].

3.1.1. Proces kreiranja Google Cloud projekta

Kreiranje Google Cloud projekta započinje prijavom u *Google Cloud Console* i kreiranjem novog projekta. Proces uključuje sljedeće korake:

- U *Google Cloud Console*, potrebno je otići na *IAM & Admin* te odabrati opciju *Create a Project*.
- U polju *Project Name*, unosi se opisni naziv projekta koji jasno definira njegovu svrhu.
- Po potrebi, moguće je urediti *Project ID*, no taj ID se ne može mijenjati nakon kreiranja projekta.
- Nakon postavljanja imena i ID-a, odabire se lokacija projekta klikom na opciju *Browse* te odabirom odgovarajuće lokacije.
- Klikom na *Create*, projekt se kreira unutar nekoliko minuta, a korisnik se preusmjerava na *Dashboard* stranicu unutar *Google Cloud Console*-a.

Ovaj proces stvara osnovnu strukturu projekta, unutar koje se može omogućiti pristup različitim Google API-ima, postaviti naplata te dodijeliti suradnike i potrebne uloge [1].

3.1.2. Omogućavanje API-ja unutar Google Cloud projekta

Jedan od ključnih koraka nakon kreiranja projekta je omogućavanje relevantnih API-ja koji će se koristiti u aplikaciji. Primjerice, za rad s *Google Calendar API*-jem, potrebno je omogućiti taj API unutar projekta. Proces uključuje sljedeće korake:

- Unutar *Google Cloud Console*-a, korisnik odabire *APIs & Services*, zatim *Library*.
- U pretraživaču API-ja, potrebno je potražiti *Google Calendar API* i kliknuti na *Enable* kako bi se omogućila integracija s aplikacijom.

Omogućavanje API-ja otvara pristup API pozivima koji omogućuju interakciju s Google-ovim servisima, kao što je dohvaćanje, kreiranje i upravljanje događajima unutar Google kalendara [1].

3.1.3. Postavljanje naplate i upravljanje korisnicima

Ovisno o vrstama API-ja koje aplikacija koristi, može biti potrebno omogućiti naplatu za Google Cloud projekt. To uključuje povezivanje projekta s računom za naplatu, koji prati korištenje API-ja te osigurava da se premašene kvote pravilno naplate. U *Billing* odjeljku unutar *Google Cloud Console*-a, korisnik može odabrati opciju *Billing & My Projects* te odabrati odgovarajući račun za naplatu [1].

Dodatno, Google Cloud projekt omogućuje dodavanje suradnika i postavljanje različitih uloga za upravljanje projektom. Unutar *IAM & Admin* postavki, mogu se dodati korisnici s različitim razinama pristupa [1].

3.2. API

Kada korisnik koristi softver kao klijent, često koristi nekakvo softversko sučelje za interakciju s softverom, ali kada je potrebno da jedan softver koristi dijelove drugog softvera tada koristimo vrstu sučelja za programiranje aplikacija (engl. *Application Programming Interface*) ili skraćeno API [2]. Ta interakcija se najčešće bazira na tome da klijent šalje HTTP zahtjev serveru na određenu lokaciju i dobiva nazad podatke. API zahtjev se sastoji od nekoliko dijelova:

- Operacija koja se izvršava (primjer. *GET, POST*)
- Autentifikacijski parametri
- Odredište - URL API krajnja točke (engl. *endpoint*)

Poziv može sadržavati i druge parametre ali ovo su tri osnovna koja će se uvijek koristiti [3].

3.2.1. HTTP zahtjevi

Kada klijent šalje zahtjev poslužitelju mora specifirati u zahtjevu koju metodu želi izvršiti, imena metoda se odnose na ono što želimo postići sa zahtjevom [4].

- **GET** metoda koristi se za dohvaćanje podataka
- **POST** - slanje i dodavanje podataka
- **PUT** - Ažuriranje podataka
- **DELETE** - brisanje resursa

3.2.2. RESTFul API

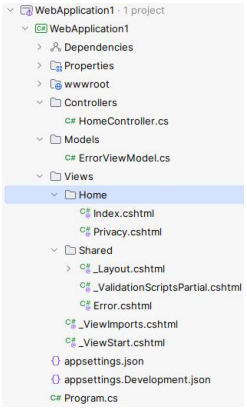
RESTFul API je vrsta API-ja koja prati REST *(*eng. representational state transfer) principe dizajna. RESTFul API može biti u bilo kojem jeziku i može koristiti bilo koju vrstu podataka [5]. Iako najčešće koristi HTTP protokol on nije nužno vezan za njega [6]. Osnovne smjernice RESTFul pristupa su:

- **Jedinstveno sučelje** - API dizajn mora biti konzistentan i predvidljiv, s pristupom resursima putem standardnih HTTP metoda kao što su GET, POST, PUT i DELETE.
- **Razdvajanje klijenta i servera** - Klijent i server su neovisni, gdje server ne čuva informacije o stanju klijenta između zahtjeva, a klijent nema direktan pristup podacima na serveru.
- **Bez stanja (engl. *Stateless*)** - Svaki zahtjev od klijenta prema serveru mora sadržavati sve potrebne informacije za obradu, bez potrebe za čuvanjem stanja na serveru.
- **Keširanje** - Resursi se mogu keširati kako bi se smanjilo opterećenje servera i omogućilo ponovnu upotrebu već preuzetih podataka.
- **Sustav slojeva** - Slojevita arhitektura omogućuje umetanje posrednika između klijenta i servera, dodajući funkcionalnosti poput keširanja ili sigurnosnih provjera.
- **Kod na zahtjev (opcionalno)** - Klijent može preuzeti i izvršiti kod od servera radi proširenja funkcionalnosti aplikacije.

3.3. ASP.NET Core

3.3.1. ASP.NET MVC

ASP.NET MVC je framework koji se bazira na MVC (engl. *Model-View-Controller*) arhitekturi, izgrađen je na .NET platformi i koristi se za izradu web aplikacija [7]. Kao što ime glasi, MVC arhitektura se sastoji od modela, pregleda (eng. *View*) i kontrolera (eng. *Controller*). Ovaj oblik dizajna prati prvi princip SOLID metoda, razdvajanja odgovornosti (eng. *Seperation of concerns*). MVC omogućava ponovnu upotrebljivost, i zbog podjele na 3 glavne komponente olakšava održavanje koda [8].



Slika 1: Prikaz MVC u ASP.NET MVC projektu (Izvor: autor)

3.3.1.1. Model

Unutar konteksta ASP.NET MVC projekta, model predstavlja C# klasu koja sadrži svojstva za spremanje podataka kojima upravljamo. Model je neovisan o korisničkom sučelju ali često će postojati pregled (engl. *View*) koji odgovara za prikaz i upravljanje modelom. Model također može sadržavati poslovnu logiku za upravljanje podacima, iako to nije učestala praksa i često se ta uloga daje servisima.

3.3.1.2. View

Pregledi (eng. *Views*) se koriste za prikazivanje podataka i korisničku interakciju. ASP.NET MVC koristi Razor stranice, sa ekstenzijom *cshtml*. Razor stranice omogućavaju pisanje C# koda unutar HTML datoteka koji služi za interaktiranje sa HTML oznakama za generiranje web sadržaja [9]. Najčešće će svaki pregled imati svoj kontroler koji je odgovoran za rad s pregledima.

3.3.1.3. Controllers

Kontroler u ASP.NET Core MVC arhitekturi služi kao posrednik između modela i prikaza. On obrađuje korisničke zahtjeve, upravlja podacima iz modela, i odlučuje koji će prikaz biti vraćen korisniku. Kontroleri su ključni dio MVC uzorka jer povezuju poslovnu logiku s korisničkim sučeljem. Kontroler je klasa koja obično nasljeđuje baznu klasu `Controller` i sadrži metode koje se nazivaju akcije (engl. *actions*). Svaka akcija odgovara određenom korisničkom zahtjevu i vraća rezultat, kao što je prikaz (*ViewResult*), JSON podaci (*JsonResult*), ili redirekcija (*RedirectResult*). Akcije također možemo opisati kao metode koje se povezivaju kad unesemo određeni URL [10].

3.3.2. ASP.NET Web API

ASP.NET Core nam omogućava da kreiramo web API s upotrebom kontrolera koji su usredočeni na resurse i primanje HTTP zahtjeva. Prednost kreiranja zasebnog Web API pro-

jekta je da ga može koristiti više različitih vrsta klijenta [11]. U ASP.NET Core možemo imati dva pristupa kreiranja API-ja:

- **API bazirani na kontrolerima (engl. *controller-based APIs*)**: U ovom pristupu, kontroleri (engl. *controllers*) su klase koje nasljeđuju `ControllerBase` klasu. Ove klase koriste se za definiranje API krajnjih točaka (engl. *endpoints*). Metode unutar kontrolera mapiraju se na određene HTTP zahtjeve (engl. *HTTP requests*, npr. GET, POST) i vraćaju odgovore u obliku JSON-a, XML-a, ili drugih formata.
- **Minimalni API-ji (engl. *minimal APIs*)**: Ovaj pristup omogućava definiranje krajnjih točaka pomoću lambda izraza (engl. *lambda expressions*) ili metoda, bez potrebe za punom klasom kontrolera. Minimalni API-ji su dizajnirani za jednostavne i brze implementacije, gdje se fokusira na definiranje krajnjih točaka uz minimalno opterećenje infrastrukture.

Za potrebe ovog rada koristit će se API bazirani na kontrolerima zbog bolje organizacije koda u cjeline te zbog jednostavnijeg rukovanja ulaznim podacima pomoću atributa *[FromBody]* i *[FromQuery]*

4. Izrada Meeting Planner aplikacije

4.1. Uvod u projekt

Prije izrade projekta trebaju se definirati funckionalnosti projekta. Cilj aplikacije je dohvatiti sve dostupne termine koje korisnik ima u svome Google kalendaru te sinkronizacija u jedan ispis koji će organizatoru sastanka prikazati kada su dostupni svi planirani korisnici i odabrana dvorana u kojoj će se sastanak održati. Na bazi toga, definiramo funkcionalnosti.

- Prijava korisnika: Korisnik se može prijaviti u sustav koristeći svoje Google podatke.
- Prikaz kalendara jednog korisnika (organizator): Korisniku se prikazuje njegov osobni kalendar s pregledom svih zakazanih sastanaka u obliku tablice.
- Upis termina u kalendare (kalendar organizatora i kalendar korisnika): Nakon pronalaska slobodnog termina, sustav će omogućiti organizatoru upis sastanka u svoj kalendar kao i njihove kalendare.
- Pronalazak slobodnih termina: Sustav pretražuje slobodne termine za sastanak uzimajući u obzir slobodno vrijeme sudionika, odabrane dane za sastanak i dostupne lokacije.

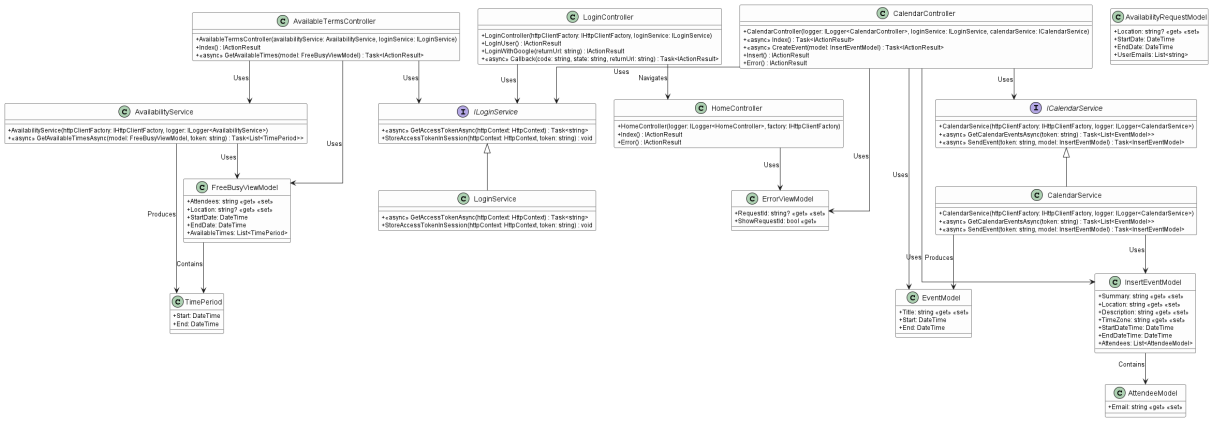
Za omogućavanje integracije s Google kalendarom koristiti ćemo Google Calendar API. RESTFul API s kojim se može komunicirati pomoću HTTP poziva.

4.2. Dizajn rješenja

U ovoj sekciji opisujemo dizajn rješenja za aplikaciju *MeetingPlanner*, koja se sastoji od klijentskih engl. *Frontend* i poslužiteljskih engl. *Backend* komponenti. UML class dijagrami koji prate ovaj opis ilustriraju strukturu ključnih klasa i njihovu međusobnu povezanost.

4.2.1. Struktura klijentskog dijela aplikacije

Na slici 2 prikazan je class dijagram za klijentski dio dio aplikacije *MeetingPlanner*. Ovaj dio aplikacije uključuje logiku za dohvaćanje slobodnih termina, autentifikaciju korisnika putem Google usluga, te rad s kalendarima korisnika.



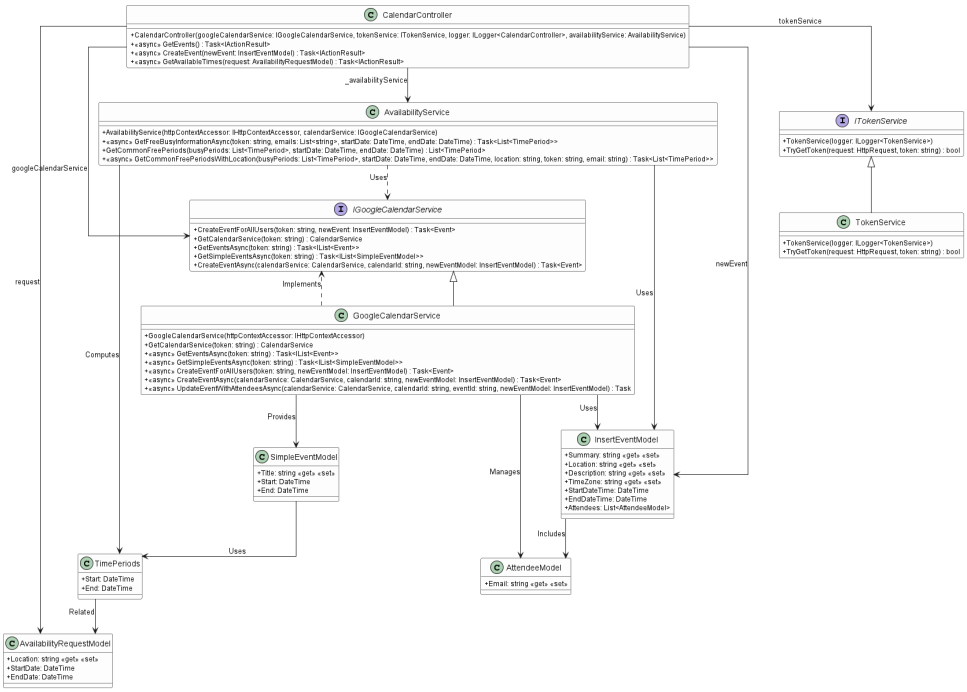
Slika 2: Class dijagram klijentskog dijela aplikacije

Na dijagramu vidimo sljedeće glavne klase i njihove odnose:

- `AvailableTermsController` koristi `AvailabilityService` kako bi dohvaćao slobodne termine korisnika. Ova klasa je odgovorna za primanje zahtjeva od korisnika i prikaz raspoloživih termina.
- `LoginController` i `LoginService` rješavaju autentifikaciju korisnika koristeći Google OAuth 2.0. `LoginService` je odgovoran za dobivanje pristupnog tokena i njegovo spremanje u sesiju.
- `CalendarController` i `CalendarService` omogućuju interakciju s Google kalendarom korisnika, uključujući kreiranje i dohvaćanje događaja.
- Modeli kao što su `FreeBusyViewModel`, `AvailabilityRequestModel`, i `InsertEventModel` definiraju strukturu podataka koja se koristi za prikaz informacija o zauzetim terminima, slanje zahtjeva i kreiranje događaja.

4.2.2. Struktura poslužiteljskog dijela

Poslužiteljsko rješenje je prikazano na slici 3, koja uključuje ključno poslovno logiku za rad s Google Calendar API-em i korisničkim autentifikacijskim tokovima.



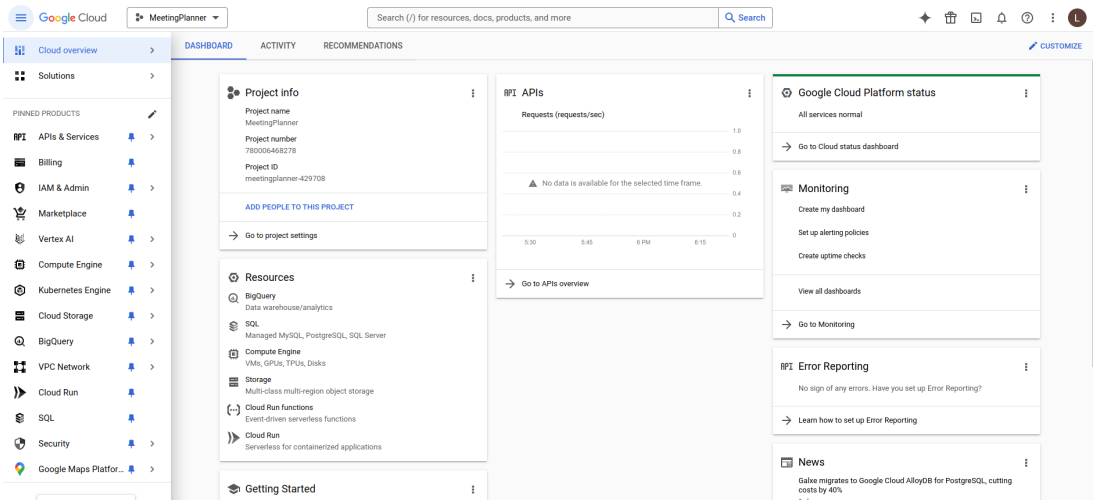
Slika 3: Class dijagram poslužiteljske komponente

Na dijagramu vidimo sljedeće klase i njihove funkcionalnosti:

- `CalendarController` je glavni kontroler koji koristi `AvailabilityService` za dohvaćanje slobodnih termina iz Google kalendara, kao i za kreiranje novih događaja.
- `AvailabilityService` koristi `IGoogleCalendarService` kako bi obavljao složene operacije nad kalendarima korisnika, poput dohvaćanja zauzetih termina i kombiniranja slobodnih perioda.
- `GoogleCalendarService` implementira stvarnu komunikaciju s Google Calendar API-jem, obavljajući pozive za dohvaćanje i ažuriranje događaja.
- Modeli kao što su `InsertEventModel`, `SimpleEventModel`, i `AvailabilityRequestModel` definiraju strukturu podataka potrebnu za komunikaciju s API-em, kao i za spremanje informacija o događajima.

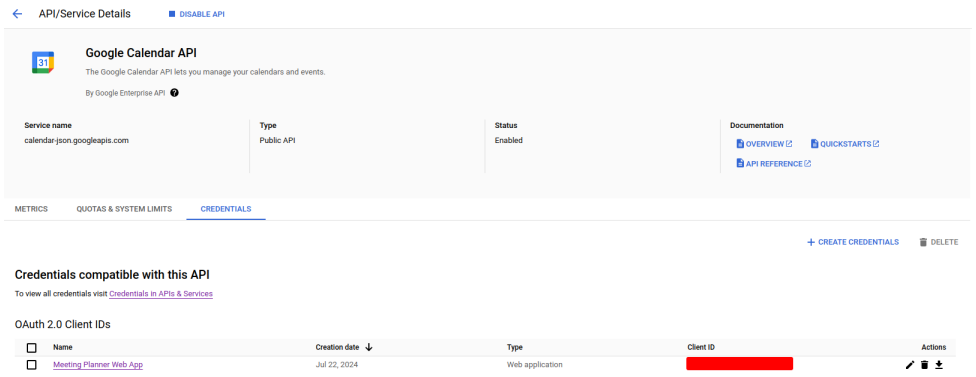
4.3. Google Workspace

Google Workspace nam omogućava platformu za integraciju Google Workspace alata kao što su Maps, Calendar ili Sheets. Za ovaj projekt su nam potrebni Google Workspace APIs da integriramo kalendar s našim rješenjem. Da možemo koristiti te servise moramo prvo stvoriti projekt u Google Cloud konzoli te omogućiti Google Calendar API za naš projekt.



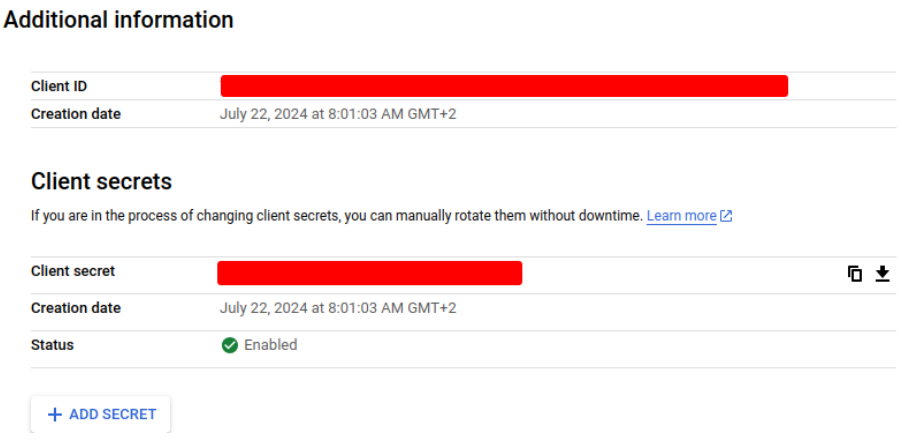
Slika 4: Google Cloud console (Izvor: autor)

Potrebno je imati aktivni Google email račun, te se prijaviti u *Google Cloud Console*. Kreiranje projekta je dosta intuitivno, najbitniji korak je omogućiti google calendar API



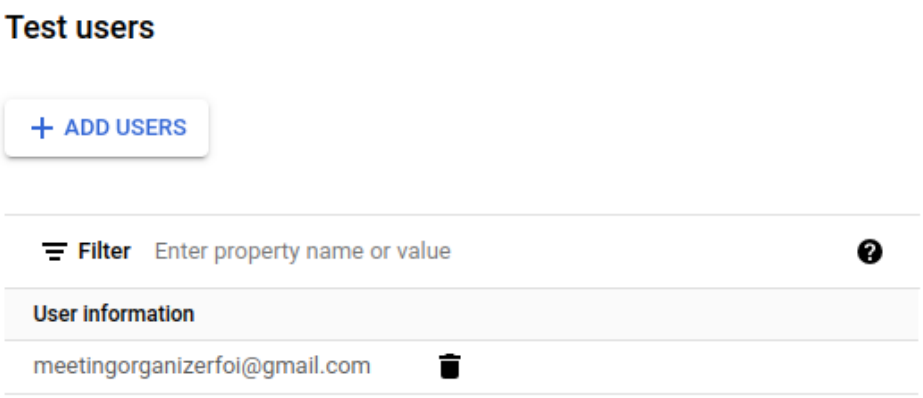
Slika 5: Omogućavanje Google Calendar API (Izvor: autor)

Generira se tajna klijenta koji nam je potreban da naša aplikacija može slati zahtjeve Google API-ju.



Slika 6: Informacije o projektu MeetingPlanner (Izvor: autor)

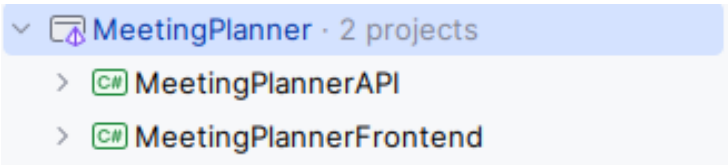
Za proces razvoja aplikacije potrebno je dodati *TestUsers* koji se mogu prijavljivati u našu aplikaciju



Slika 7: Informacije o projektu MeetingPlanner (Izvor: autor)

4.4. Stvaranje projekta

Za stvaranje projekta prvo je potrebno stvoriti prazno rješenje (engl. *Solution*), te kreirati zasebni WEB API i MVC projekt.



Slika 8: Rješenje MeetingPlanner (Izvor: autor)

Za dodatnu konfiguraciju potrebno je preuzeti json datoteku koja sadrži podatke koji našoj aplikaciji omogućuju pozivanje google servisa.

- GoogleId (Identifikator klijenta): jedinstveni identifikator dodijeljen aplikaciji od strane Googlea. Identificira vašu aplikaciju na Googleovim poslužiteljima prilikom slanja zahtjeva, kao što su autentifikacija korisnika ili pristup API-jima.
- GoogleSecret (Tajni ključ): povjerljivi ključ povezan s vašom Google aplikacijom. Koristi se zajedno s GoogleId za autentifikaciju vaše aplikacije prema Googleu.

4.5. Autentifikacija

Kako bi korisnik aplikacije mogao pristupiti vanjskim uslugama on mora biti prijavljen, korisnik će se prijavljivati putem svojeg Google računa koristeći OAuth 2.0 protokol. Prvo je potrebno konfigurati autentifikaciju projekta u *Program.cs* datoteci.

```
builder.Services.AddAuthentication( configureOptions: options =>
{
    options.DefaultAuthenticateScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    options.DefaultSignInScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = GoogleDefaults.AuthenticationScheme;
})
.AddCookie()
.AddGoogle(options =>
{
    var googleAuthNSection = builder.Configuration.GetSection( key: "Authentication:Google");
    var clientId :string? = googleAuthNSection["ClientId"];
    var clientSecret :string? = googleAuthNSection["ClientSecret"];

    if (string.IsNullOrEmpty(clientId) || string.IsNullOrEmpty(clientSecret))
    {
        throw new InvalidOperationException("Google ClientId and ClientSecret must be provided.");
    }

    options.ClientId = clientId;
    options.ClientSecret = clientSecret;
    options.Scope.Add("https://www.googleapis.com/auth/calendar");
    options.CallbackPath = "/signin-google";
    options.SaveTokens = true;
    options.Scope.Add("openid");
    options.Scope.Add("profile");
    options.Scope.Add("email");
});
```

Slika 9: Postavljanje autentifikacije (Izvor: autor)

Kod konfigurira autentifikaciju postavljanjem kolačića za prijavu i koristeći Google za autentifikaciju korisnika, uključujući konfiguraciju za Google kalendar, pohranu OAuth tokena, i određivanje putanje za povratak korisnika nakon prijave. Kada je postavljena konfiguracija kreiramo login stranicu, s obzirom da koristimo Google autentifikaciju kreiramo element koji poziva akciju *LoginWithGoogle*.

```
public IActionResult LoginWithGoogle(string returnUrl = "/")
{
    var properties = new AuthenticationProperties
    {
        RedirectUri = Url.Action(nameof(Callback), new { returnUrl })
    };
    return Challenge(properties, params authenticationSchemes: GoogleDefaults.AuthenticationScheme);
}
```

Slika 10: Metoda LoginWithGoogle (Izvor: autor)

Metoda *LoginWithGoogle* inicijalizira objekt *AuthenticationProperties* s postavkom *RedirectUri* koja određuje putanju na metodu *Callback* nakon prijave putem Googlea. Metoda `Challenge` pokreće izazov za autentifikaciju koristeći Google kao pružatelja, preusmjeravajući korisnika na Google za prijavu ako nije već prijavljen. Nakon uspješne prijave, korisnik će biti vraćen na metodu `Callback` gdje se token sprema u sesiju što nam omogućava autorizirane API pozive.

```
public async Task<IActionResult> Callback(string code, string state,string returnUrl = "/")
{
    try
    {
        var token :string = await _loginService.GetAccessTokenAsync(HttpContext);
        _loginService.StoreAccessTokenInSession(HttpContext, token);
        Console.WriteLine($"Token length: {token.Length}");
        Console.WriteLine("Token stored in session.");
        return RedirectToAction("Index", controllerName: "Home");
    } catch (UnauthorizedAccessException ex)
    {
        Console.WriteLine(ex.Message);
        TempData["ErrorMessage"] = "Error logging in: " + ex.Message;
        return Unauthorized(ex.Message);
    }
}
```

Slika 11: Metoda Callback (Izvor: autor)

Za spremanje podataka u sesiju definira se metoda *StoreAccessTokenInSession*. Metoda se nalazi u zasebnom login servisu unutar kojega se također nalazi metoda za dohvaćanje tokena za kasniju upotrebu *GetAccessTokenAsync*.

```
public async Task<string> GetAccessTokenAsync(HttpContext httpContext)
{
    var authenticateResult = await httpContext.AuthenticateAsync(CookieAuthenticationDefaults.AuthenticationScheme);

    if (!authenticateResult.Succeeded)
    {
        throw new UnauthorizedAccessException("Authentication failed.");
    }


    var token :string? = await httpContext.GetTokenAsync("access_token");

    if (string.IsNullOrEmpty(token))
    {
        throw new UnauthorizedAccessException("Access token not found.");
    }

    return token;
}

0+1 usages  LeoCavar
public void StoreAccessTokenInSession(HttpContext httpContext, string token)
{
    httpContext.Session.SetString(AccessTokenKey, token);
}
```

Slika 12: Login servis (Izvor: autor)

4.6. Prikaz događaja

Za prikaz događaja implementiramo zaseban pregled eng. *View*. Koristiti će se mogućnost razor stranica da se upisuje C# kod i definirat će se jednostavna provjera varijable *isLoggedIn*.

```
@model IEnumerable<EventModel>

@{
    ViewData["Title"] = "Calendar View";
    var isLoggedIn = ViewData["IsLoggedIn"] as bool? ?? false;
}
<br />
<h2 class="text-center">Google Calendar Events</h2>

@if (isLoggedIn)
{
    <table class="table">
        <thead>
            <tr>
                <th>Description</th>
                <th>Start</th>
                <th>End</th>
            </tr>
        </thead>
        <tbody>
            @foreach (var eventItem :EventModel in Model)
            {
                <tr>
                    <td>@eventItem.Title</td>
                    <td>@eventItem.Start</td>
                    <td>@eventItem.End</td>
                </tr>
            }
        </tbody>
    </table>
} else
{
    <h3 class="text-center">No events found. Try logging in.</h3>
}
```

Slika 13: Pregled kalendara (Izvor: autor)

Definira se klasa *EventModel* koja sadrži naslov, vrijeme početka i vrijeme kraja događaja. Klasa služi za mapiranje događaja koje dobivamo pozivom API-ju u korisniku lako čitljiv oblik. Nakon definiranja ispisa događaja, potrebno je implementirati logiku unutar kontrolera i servisa. Kontroler nam služi za obradu zahtjeva dok se sva poslovna logika i interakcija s podacima izvršava u servisu *CalendarService*.

```
public async Task<IActionResult> Index()
{
    string token;
    try
    {
        token = await _loginService.GetAccessTokenAsync(HttpContext);
    } catch (UnauthorizedAccessException ex)
    {
        _logger.LogInformation(ex.Message);
        TempData["ErrorMessage"] = "Login error: " + ex.Message.ToString();
        ViewData["IsLoggedIn"] = false;
        return View();
    }

    var events :List<EventModel> = await _calendarService.GetCalendarEventsAsync(token);

    if (events == null)
    {
        TempData["ErrorMessage"] = "Error retrieving data.";
        return StatusCode(500, "Unable to retrieve data.");


    }

    ViewData["IsLoggedIn"] = true;
    var model :List<EventModel> = events.ToList<EventModel>();
    return View(model);
}
```

Slika 14: Pregled kalendara kontroler (Izvor: autor)

Kontroler dohvaća token pomoću funkcije *GetAccessTokenAsync* te poziva metodu *GetCalendarEventsAsync* koristeći token da pošalje poziv API-ju te upravlja greškama koristeći try/catch metode. Unutar *CalendarService* metoda *GetCalendarEventsAsync* stvara klijent pod imenom "API", to je klijent koji sadrži putanju na poslužitelja koji sadrži API krajnja točke. Zahtjevu se dodaje token u *AuthenticationHeaderValue* da se mogu dohvatiti podaci i šalje se zahtjev na definiranu krajnja točku "*calendar/events*".

```
var httpClient = _httpClientFactory.CreateClient( name: "API");
httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue( scheme: "Bearer", token);

var response = await httpClient.GetAsync( requestUri: "calendar/events");
```

Slika 15: Slanje zahtjeva na API krajnja točku za dohvaćanje kalendara (Izvor: autor)

Odgovor se mapira iz JSON stringa u .NET objekt, u ovom kontekstu se pretvara u *EventModel*.

```
try
{
    var calendarData :string = await response.Content.ReadAsStringAsync();
    _logger.LogInformation($"Calendar Data: {calendarData}");
    return JsonConvert.DeserializeObject<List<EventModel>>(calendarData)!;
} catch (Exception ex)
{
    _logger.LogError(ex, message: "Error parsing calendar data.");
    return null;
}
```

Slika 16: Metoda za upravljanjem povratnog JSON teksta (Izvor: autor)

Ovo je sva potrebna logika za klijentski dio sustava. Potrebno je implementirati poslužiteljski dio. Poslužiteljski dio se implementira tako što se definira *ApiController* ruta koja se pozvala u metodi *GetCalendarEventsAsync*. Prvobitno se izvršava validacija tokena te se dohvaćaju događaji pomoću metode *GetSimpleEventsAsync* i provjerava se uspješnost operacije.

```
[HttpGet( template: "events")]
LeoCavar +1
public async Task<IActionResult> GetEvents()
{
    if (!_tokenService.TryGetToken(Request, out var token :string ))
    {
        return Unauthorized(new { message = "Authorization header or token is missing." });
    }

    var events :IList<SimpleEventModel> = await _googleCalendarService.GetSimpleEventsAsync(token);

    if (events == null)
    {
        return StatusCode(500, "Unable to retrieve data.");
    }

    return Ok(events);
}
```

Slika 17: Kalendar "events" krajnja točka (Izvor: autor)

Metoda za dohvaćanje se sastoji od dva dijela, prvi dio je stvaranje instance *CalendarService*. Ovaj servis je od GoogleAPI-ja te nam omogućava manipulaciju podacima autoriziranog korisnika unutar google kalendara.

```
public CalendarService GetCalendarService(string token)
{
    if (string.IsNullOrEmpty(token))
    {
        Console.WriteLine("Access token was not found or is null.");
        throw new UnauthorizedAccessException("No access token found.");
    }

    var credential = GoogleCredential.FromAccessToken(token)
        .CreateScoped(CalendarService.Scope.Calendar);

    return new CalendarService(new BaseClientService.Initializer
    {
        HttpClientInitializer = credential,
        ApplicationName = "MeetingPlanner"
    });
}
```

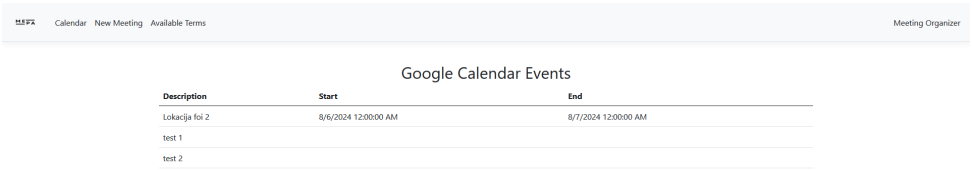
Slika 18: Metoda za dohvaćanje instance Google kalendar servisa (Izvor: autor)

Drugi dio procesa je dohvaćanje *Events.List* resursa i mapiranje u obliku *SimpleEvent-Model*.

```
public async Task<IList<Event>> GetEventsAsync(string token)
{
    var calendarService = GetCalendarService(token);
    var calendarId = "primary";

    var request = calendarService.Events.List(calendarId);
    request.ShowDeleted = false;
    request.SingleEvents = true;
    request.OrderBy = EventsResource.ListRequest.OrderByEnum.StartTime;

    var events = await request.ExecuteAsync();
    return events.Items;
}

0+1 usages  LeoCavar
public async Task<IList<SimpleEventModel>> GetSimpleEventsAsync(string token)
{
    var events :IList<Event> = await GetEventsAsync(token);
    var simpleEvents :List<SimpleEventModel> = events.Select(e :Event => new SimpleEventModel
    {
        Title = e.Summary,
        Start = e.Start?.DateTime,
        End = e.End?.DateTime
    }).ToList();

    return simpleEvents;
}
```

Slika 19: Metoda GetSimpleEventAsync (Izvor: autor)

Nakon implementacije metode naša funkcionalnost je potpuna, te se kalendar vraća u obliku tablice prijavljenom korisniku.



Slika 20: Prikaz kalendara na web stranici (Izvor: autor)

4.6.1. Protok podataka

Protok podataka u ovoj aplikaciji je jasno definiran za sve funkcionalnosti koje će se implementirati dalje u ovome radu s malim razlikama ovisno o funkcionalnosti ali uzorak će ostati isti.

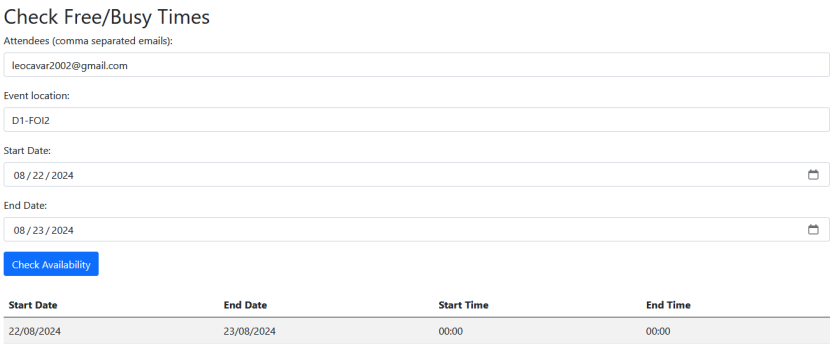
- **Klijentov zahtjev**: Korisnik putem web aplikacije odabire funkcionalnost koju želi koristiti putem korisničkoj sučelja te se šalje odgovarajući zahtjev
- **Obrada zahtjeva na poslužitelju**: Poslužiteljski kontroler prima zahtjev i prosljeđuje ga odgovarajućem servisu.
- **Autentifikacija i autorizacija**: Prilikom dohvaćanja podataka iz Google Calendar API-ja, aplikacija koristi OAuth 2.0 autentifikaciju. Token za pristup provjerava se i koristi za autentifikaciju zahtjeva prema Googleovim uslugama.
- **Odgovor poslužitelja**: Nakon obrade podataka, poslužitelj šalje odgovor klijentu. Ovaj odgovor uključuje popis slobodnih termina za sve korisnike u traženom vremenskom periodu, koji se klijentu vraćaju u JSON formatu ili potvrdu o izmjeni i unosu podataka.
- **Prikaz rezultata klijentu**: Klijentska aplikacija prima JSON odgovor i prikazuje slobodne termine korisniku u vizualno preglednom formatu, omogućujući mu da jednostavno vidi kada su svi korisnici dostupni ili se prikazuje odgovarajuća greška ovisno o odgovoru.

4.7. Pronalazak slobodnih termina

Kako Google Calendar API ne omogućuje izravno dohvaćanje slobodnih termina, postupak pronalaska slobodnih perioda obuhvaća sljedeće korake:

- Dohvat kalendara svih dolaznika (engl. *Attendees*)
- Filtriranje kalendara po lokaciji
- Identificiranje zauzetih termina unutar zadanog vremenskog raspona
- Kombiniranje svih zauzetih termina kako bi se pronašli zajednički slobodni periodi
- Oduzimanje zauzetih termina od cijelog vremenskog raspona kako bi se identificirali slobodni termini

Korisničko sučelje uključuje obrazac koji šalje zahtjev na *calendar/get-available-times* nakon unosa podataka. Odgovor se prikazuje u tablici sa slobodnim periodima.



Slika 21: Web sučelje za prikaz dostupnih termina (Izvor: autor)

Funkcija *GetAvailableTimesAsync* vraća popis objekata *TimePeriod* koji sadrži vrijeme početka i kraja slobodnog perioda. Zahtjev uključuje token u headeru, a u tijelu se nalaze vremenski period, lokacija i email adrese korisnika. Funkcija šalje POST zahtjev na *get-available-times* krajnju točku, koji zatim prosljeđuje zahtjev metodi *GetCommonFreePeriodsWithLocation* nakon validacije tokena i zahtjeva. Unutar akcije *GetAvailableTimes* na koju se šalje zahtjev dohvaćaju se prvo svi termini kada su korisnici zauzeti.

```
public async Task<List<TimePeriod>> GetFreeBusyInformationAsync(string token, List<string> emails, DateTime startDate, DateTime endDate)
{
    var calendarService = _calendarService.GetCalendarService(token);

    var request = new FreeBusyRequest
    {
        TimeMin = startDate,
        TimeMax = endDate,
        Items = emails.Select(email:string => new FreeBusyRequestItem { Id = email }).ToList()
    };

    var freeBusyRequest = calendarService.Freebusy.Query(request);
    var response = await freeBusyRequest.ExecuteAsync();

    var busyPeriods:List<TimePeriod> = response.Calendars.Values.SelectMany(calendar => calendar.Busy).Select(b:TimePeriod => new TimePeriod
    {
        Start = (DateTime)b.Start,
        End = (DateTime)b.End
    }).ToList();

    return busyPeriods;
}
```

Slika 22: Metoda *GetFreeBusyInformationAsync* (Izvor: autor)

Nakon dohvaćanja zauzetih termina za sve korisnike, potrebno je preuzeti termine kada je dvorana zauzeta.

```
try
{
    var freeBusyRequest = calendarService.Freebusy.Query(request);
    var response = await freeBusyRequest.ExecuteAsync();

    var locationBusyPeriods = new List<TimePeriod>();

    foreach (var calendar in response.Calendars.Values)
    {
        var eventDetailsRequest = calendarService.Events.List(email);
        eventDetailsRequest.TimeMin = startDate;
        eventDetailsRequest.TimeMax = endDate;
        var events = await eventDetailsRequest.ExecuteAsync();

        foreach (var eventItem in events.Items)
        {
            if (eventItem.Location != null && eventItem.Location.Contains(location, StringComparison.OrdinalIgnoreCase))
            {
                if (eventItem.Start.DateTime.HasValue && eventItem.End.DateTime.HasValue)
                {
                    locationBusyPeriods.Add(new TimePeriod
                    {
                        Start = eventItem.Start.DateTime.Value,
                        End = eventItem.End.DateTime.Value
                    });
                }
            }
        }
    }
```

Slika 23: Dohvaćanje zauzetih termina odabrane lokacije (Izvor: autor)

Nakon preuzimanja termina lokacije imamo sve potrebne podatke, prvo je potrebno spojiti zauzete periode korisnika s zauzetim terminima dvorane, metoda *Concat* omogućava da spojimo periode u jednu varijablu *combinedBusyPeriods*. Nakon spajanja u jednu varijablu, moramo pretvoriti popis zauzetih termina u popis slobodnih termina. Implementira se metoda *GetCommonFreePeriods* i kao povratna vrijednost klijentu se vraćaju svi slobodni periodi.

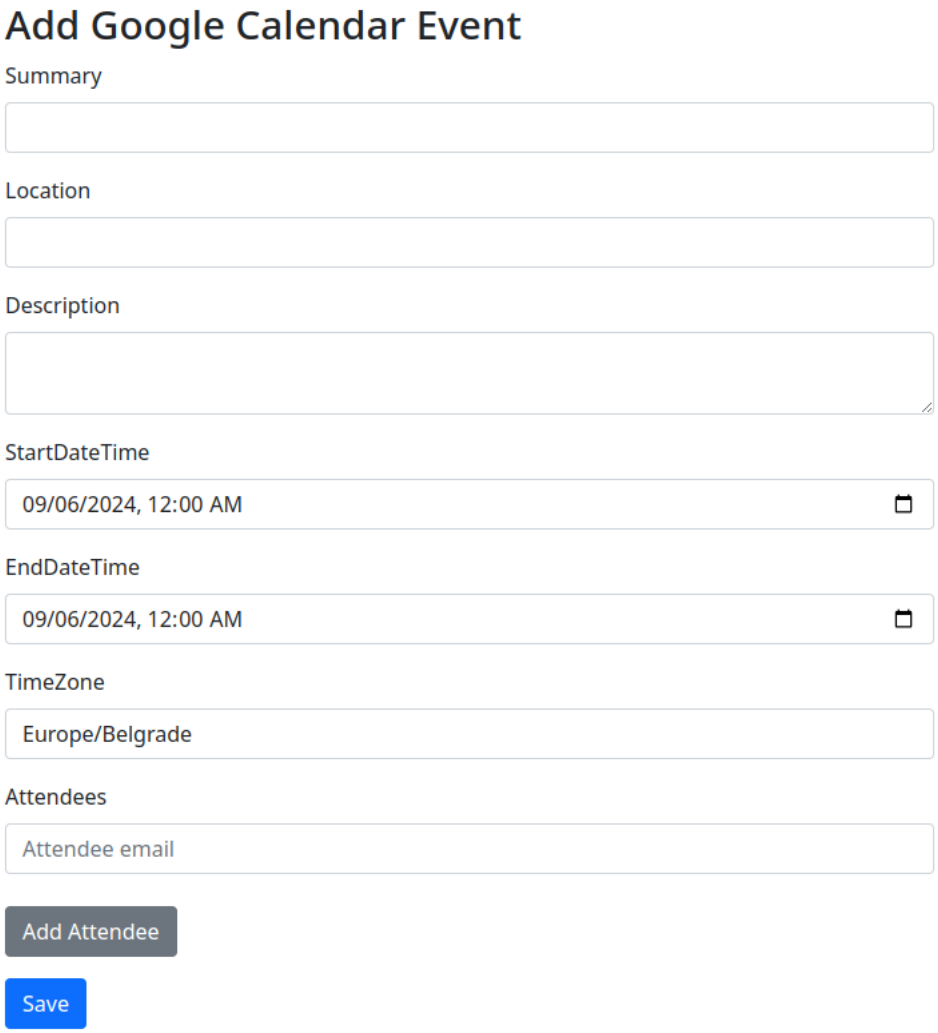
```
public List<TimePeriod> GetCommonFreePeriods(List<TimePeriod> busyPeriods, DateTime startDate, DateTime endDate)
{
    var commonFreePeriods = new List<TimePeriod>();

    // Initialize free periods with the full range
    var freePeriods = new List<TimePeriod> { new TimePeriod { Start = startDate, End = endDate } };

    // Subtract busy periods
    freePeriods = SubtractBusyPeriods(freePeriods, busyPeriods);

    return freePeriods;
}

1 usage  & LeoCavar
private List<TimePeriod> SubtractBusyPeriods(List<TimePeriod> freePeriods, List<TimePeriod> busyPeriods)
{
    var newFreePeriods = new List<TimePeriod>();

    foreach (var freePeriod in freePeriods)
    {
        var currentFreeStart :DateTime? = freePeriod.Start;

        foreach (var busyPeriod in busyPeriods)
        {
            if (busyPeriod.Start >= freePeriod.End)
                break;

            if (busyPeriod.End <= currentFreeStart)
                continue;

            if (busyPeriod.Start > currentFreeStart)
                newFreePeriods.Add(new TimePeriod { Start = currentFreeStart, End = busyPeriod.Start });

            currentFreeStart = busyPeriod.End > freePeriod.End ? freePeriod.End : busyPeriod.End;
        }

        if (currentFreeStart < freePeriod.End)
            newFreePeriods.Add(new TimePeriod { Start = currentFreeStart, End = freePeriod.End });
    }

    return newFreePeriods;
}
```

Slika 24: Metode za pretvaranje zauzetih termina u slobodne (Izvor: autor)

Metode *GetCommonFreePeriods* i *SubtractBusyPeriods* instanciraju popis *TimePeriod* unutar odabranog vremenskog raspona te pomoću metode *SubtractBusyPeriods* se od slobodnih vremena oduzimaju zauzeta vremena, stvarajući popis u kojemu se nalaze svi slobodni termini korisnika.

4.8. Unos sastanka

Kao završni dio aplikacije potrebno je omogućiti korisniku da unosi aktivnosti u kalendare korisnika. Prvo je potrebno kreirati sučelje za unos, korisnik može unositi opis, lokaciju sastanka, opis , vremenski period aktivnosti, vremensku zonu i goste sastanka.



Slika 25: Sučelje za unos sastanka (Izvor: autor)

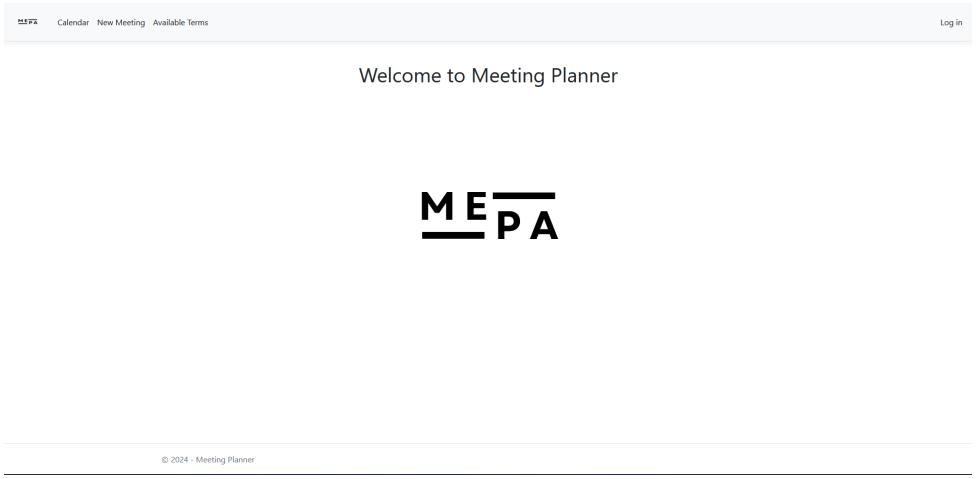
Potrebno je poslati POST zahtjev s svim parametrima, uneseni podaci se mapiraju u klasu *InsertEventModel* te se šalju na *calendar/events* krajnju točku. Stvara se novi *Event* objekt u kalendaru organizatora, te se na taj događaj dodaju članovi.

```
public async Task<Event> CreateEventForAllUsers(string token, InsertEventModel newEventModel)
{
    var calendarService = GetCalendarService(token);
    var primaryCalendarId = "primary";


    // Create the event in the organizer's calendar
    var createdEvent = await CreateEventAsync(calendarService, primaryCalendarId, newEventModel);

    // Update the event to include all attendees
    var updatedEventModel = new InsertEventModel
    {
        Summary = newEventModel.Summary,
        Location = newEventModel.Location,
        Description = newEventModel.Description,
        StartDateTime = newEventModel.StartDateTime,
        EndDateTime = newEventModel.EndDateTime,
        TimeZone = newEventModel.TimeZone,
        Attendees = newEventModel.Attendees
    };

    await UpdateEventWithAttendeesAsync(calendarService, primaryCalendarId, createdEvent.Id, updatedEventModel);

    return createdEvent;
}
```

Slika 26: Implementacija unosa događaja u kalendare korisnika (Izvor: autor)

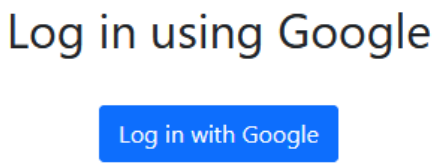
5. Demonstracija korištenja aplikacije

Kada korisnik dođe na stranicu prvo se prezentira poruka pozdrava i navigacija.



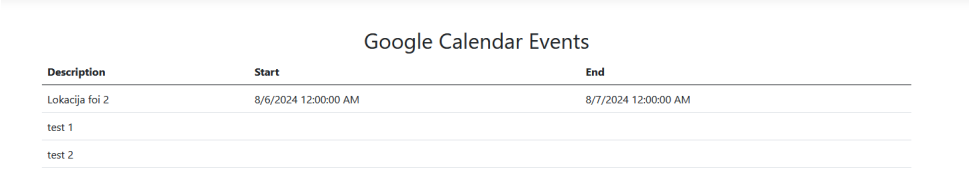
Slika 27: Početna stranica (Izvor: autor)

Prije nego što može koristiti funkcionalnosti aplikacije korisnik se mora prijaviti sa Google računom.



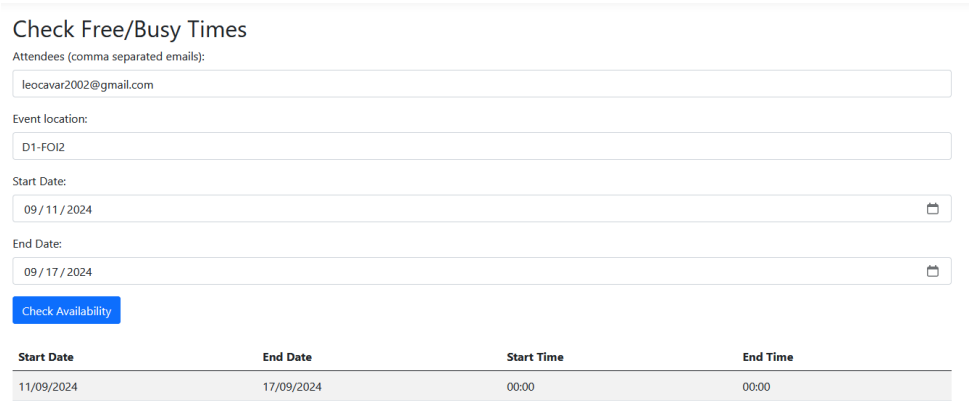
Slika 28: Login stranica (Izvor: autor)

Kada ode na stranicu *Calendar* korisniku se prikazuju njegovi događaji u kalendaru



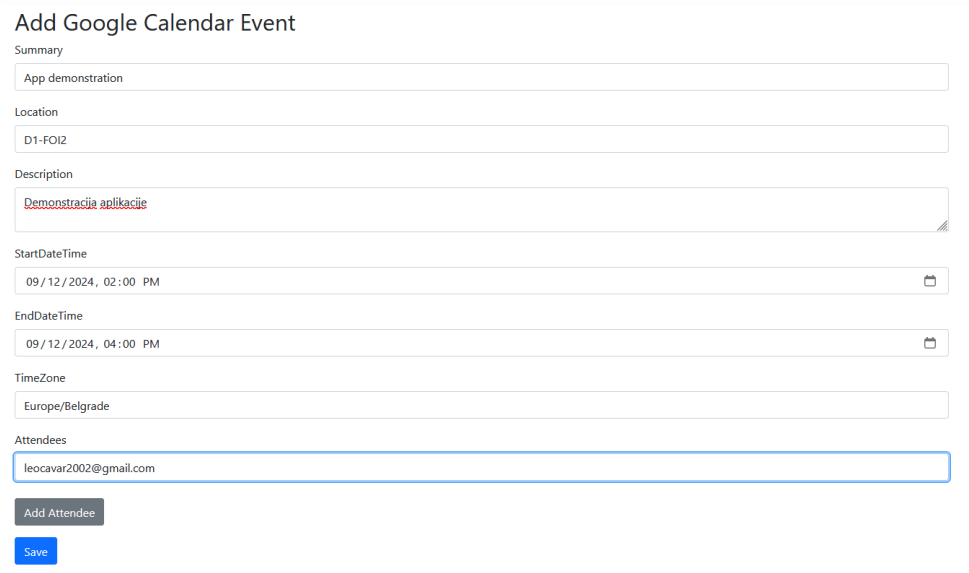
Slika 29: Kalendar prijavljenog korisnika (Izvor: autor)

Korisnik uzima u obzir svoje termine te provjerava kalendare korisnika. Unosi podatke o korisniku za kojeg se provjerava termin te se automatski ulogirani korisnik dodaje u provjeru i dobiva povratnu informaciju o slobodnim terminima.



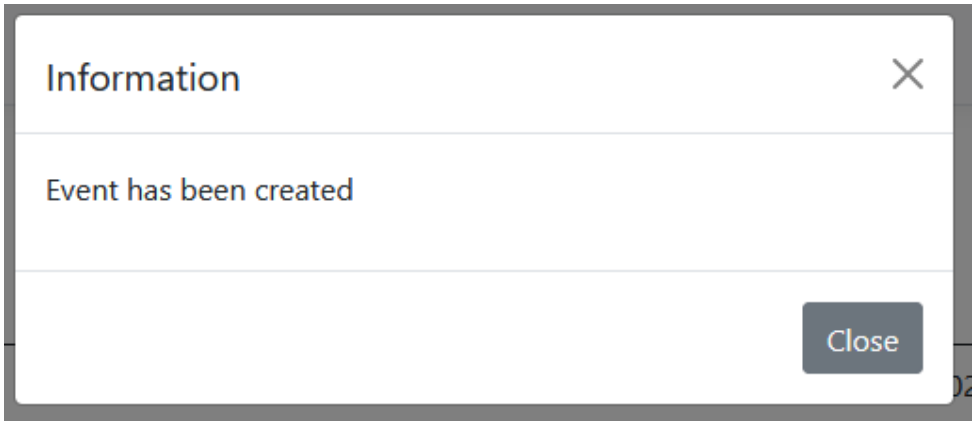
Slika 30: Provjera dostupnosti korisnika (Izvor: autor)

Uvidom u kalendar gostiju sastanka korisnik ima opciju dodati novi sastanak u kalendare korisnika. Korisnik unosi odgovarajuće podatke te unosi sastanak u kalendare svih korisnika.



Slika 31: Korisničko sučelje za unos sastanka (Izvor: autor)

Korisniku se daje odgovarajuća poruka ako je sastanak uspješno unesen.



Slika 32: Potvrda o unosu sastanka (Izvor: autor)

Kada gosti sastanka otvore svoj google kalendar vide sastanak koji smo unjeli u njegov kalendar te je taj događaj prisutan i u kalendaru organizatora.

Slika 33: Google kalendar gosta sastanka (Izvor: autor)

6. Zaključak

Zaključno, ovaj rad prikazuje proces izrade aplikacije za planiranje sastanaka korištenjem Google Calendar API-ja, razvijene u ASP.NET Core tehnologiji. Kroz detaljno objašnjenje, prikazane su glavne funkcionalnosti aplikacije kao što su autentifikacija korisnika, dohvaćanje događaja iz Google kalendara te prikaz i manipulacija tim događajima. Rad se također osvrće na tehničke aspekte implementacije poput uporabe RESTful API-ja i MVC arhitekture, čime se naglašava važnost modularnosti i razdvajanja odgovornosti u softverskom razvoju. Korištenje Google Workspace API-ja omogućilo je integraciju vanjskih servisa. Aplikacija razvijena u sklopu ovog rada pokazuje kako se napredne tehnike web razvoja mogu iskoristiti za rješavanje konkretnih poslovnih izazova u kontekstu organizacije sastanaka. Izvorni kod i kompletan projekt dostupan je na sljedećem linku: https://github.com/LeoCavar/MeetingPlanner

Popis literature

[1] *Create a Google Cloud project*, https://developers.google.com/workspace/guides/create-project (Pristupano: 6.9.2024.), 2024.

[2] M. Biehl, *API Architecture* (API-University Series). CreateSpace Independent Publishing Platform, 2015., ISBN: 9781508676645. adresa: https://books.google.ba/books?id=6D64DwAAQBAJ.

[3] 3. AltexSoft, *"What is API: Definition, Types, Specifications, Documentation". altexsoft*, https://www.altexsoft.com/blog/what-is-api-definition-types-specifications-documentation/ (Pristupano: 31.7.2024.)

[4] R. Maurya, K. A. Nambiar, P. Babbe, J. P. Kalokhe, Y. S. Ingle i N. F. Shaikh, *Application of Restful APIs in IOT: A Review*, https://www.ijraset.com (Pristupano: 9.8.2024.), 2021.

[5] IBM, *What is a REST API?* https://www.ibm.com/topics/rest-apis (Pristupano: (1.8.2024)).

[6] Microsoft, *RESTful web API design*, https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design (Pristupano: 1.8.2024.), 2023.

[7] C. Tyler, *ASP.NET MVC Tutorial for Beginners: What is, Architecture*, https://www.guru99.com/asp-net-mvc-tutorial.html (Pristupano: 12.7.2024.), 2024.

[8] GeeksforGeeks, *MVC Design Pattern*, https://www.geeksforgeeks.org/mvc-design-pattern/ (Pristupano: 12.7.2024.), veljača 2024.

[9] S. Smith i D. Brock, *Views in ASP.NET Core MVC*, https://learn.microsoft.com/en-us/aspnet/core/mvc/views/overview?view=aspnetcore-6.0 (Pristupano: 12.7.2024.), 2022.

[10] S. Walther, *ASP.NET MVC Controller Overview (C#)*, https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/controllers-and-routing/aspnet-mvc-controllers-overview-cs (Pristupano: 12.7.2024.), 2022.

[11] A. autori, *Izbor između API-ja baziranih na kontrolerima i minimalnih API-ja*, https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-6.0 (Pristupano: 12.7.2024.), 2023.

Popis slika