

**SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN**

Leo Čavar

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 Čavar

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 Čavar

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 se obrađuje korištenje .NET tehnologija za izradu programa za dogo-
varanje sastanka. Kroz rad se obrađuje kratko o ASP. NET Core i ASP. NET Web API tehno-
logijama, kao i koncepte API-ja i HTTP metoda, Google API-ja, uključujući Google Calendar
API, autentifikaciju i autorizaciju korisnika pomoću OAuth 2.0 protokola, te primjere zahtjeva za
dohvaćanje i upravljanje kalendarskim događajima kroz implementaciju .NET web aplikacije

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

Sadržaj

1. Uvod	1
2. Metode i tehnike rada	2
3. API	3
3.1. HTTP zahtjevi	3
3.2. RESTFul API	3
4. ASP.NET Core	5
4.1. ASP.NET MVC	5
4.1.1. Model	6
4.1.2. View	6
4.1.3. Controllers	6
4.2. ASP.NET Web API	6
5. Praktični rad: Meeting Planner	8
5.1. Uvod u projekt	8
5.2. Google Workspace	8
5.3. Stvaranje projekta	9
5.4. Autentifikacija	10
5.5. Prikaz događaja	12
5.5.1. Protok podataka	16
5.6. Pronalazak slobodnih termina	16
5.7. Unos sastanka	19
6. Zaključak	20
Popis literature	21
Popis slika	22

1. Uvod

U ovom radu ćemo se usredotočiti na izradu programa za dogovaranje sastanaka kroz upotrebu Google API-ja, koji nam omogućava interakciju s iznimno popularnim Google kalendarom. Realizirat ćemo program koristeći ASP.NET Core i Razor stranice za izradu front-end sučelja te upravljanje podacima, dok će ASP.NET Web API služiti za primanje HTTP zahtjeva i komuniciranje s Google servisima za manipuliranje događajima.

2. Metode i tehnike rada

Za pisanje teksta i formatiranje ovog rada koristio se LaTeX unutar programa Visual Studio Code. Za izradu praktičnog dijela korišten je Visual Studio 2022 i JetBrains Rider.

3. 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.[1] Ta interakcija se najčešće bazira na tome da klijent šalje HTTP zahtjev serveru na određenu lokaciju i dobivaju se nazad podaci. API zahtjev se sastoji od nekoliko dijelova [2]

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

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

3.1. HTTP zahtjevi

Kada klijent šalje zahtjev poslužitelju mora specificirati u zahtjevu koju metodu želi izvršiti, imena metoda se odnose na ono što želimo postići sa zahtjevom. [3]

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

3.2. RESTFul API

RESTFul API je vrsta API-ja koja prati REST (eng. *representational state transfer*) principe dizajna, može biti u bilo kojem jeziku i može koristiti bilo koju vrstu podataka [4]. Iako najčešće koristi HTTP protokol on nije nužno vezan za njega.[5]

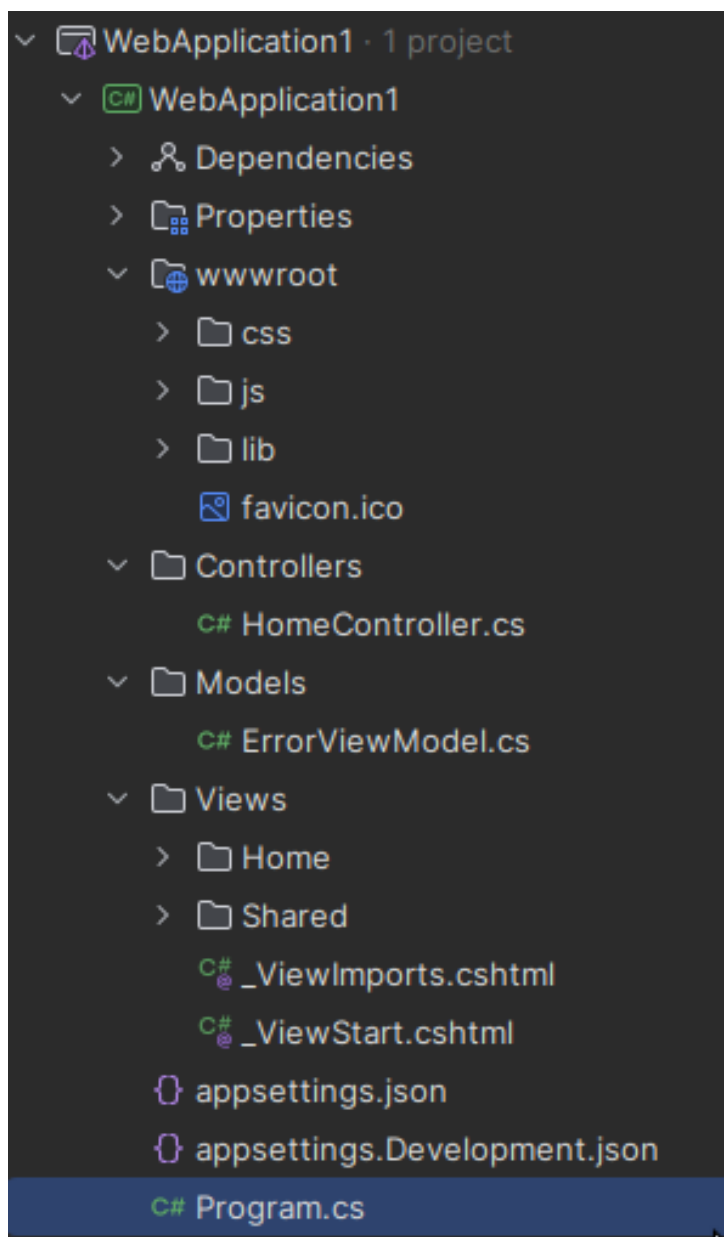
- **Jedinstveno sučelje** - API dizajn mora biti konzistentan i predvidljiv, s pristupom resursima putem standardnih HTTP metoda kao što su GET, POST, PUT i DELETE.
- **Razdvajanje klijenta i servera** - Klijent i server su neovisni, gdje server ne čuva informacije o stanju klijenta između zahtjeva, a klijent nema direktan pristup serverovim podacima.
- **Bezustanje (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.
- **Kôd na zahtjev (opcionalno)** - Klijent može preuzeti i izvršiti kod od servera radi proširenja funkcionalnosti aplikacije.

4. ASP.NET Core

4.1. 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 [6]. 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. [7]



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

4.1.1. 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. *textView*) 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.

4.1.2. View

Pregledi (eng. *Views*) se koriste za prikazivanje podataka i korisničku interakciju. ASP.NET MVC koristi Razor stranice, sa ekstenzijom *csharp*. Razor stranice omogućavaju pisanje C# koda unutar HTML datoteka koji služi za interaktiranje sa HTML oznakama za generiranje web sadržaja [8]. Najčešće će svaki pregled imati svoj kontroler koji je odgovoran za rad s pregledima.

4.1.3. 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. [9]

4.2. ASP.NET Web API

ASP.NET Core nam omogućava da kreiramo web API s upotrebom kontrolera koji su usredotočeni na resurse i primanje HTTP zahtjeva. Prednost kreiranja zasebnog Web API projekta je da ga može koristiti više različitih vrsta klijenta.[10] 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 implemen-

tacije, 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]*

5. Praktični rad: Meeting Planner

5.1. Uvod u projekt

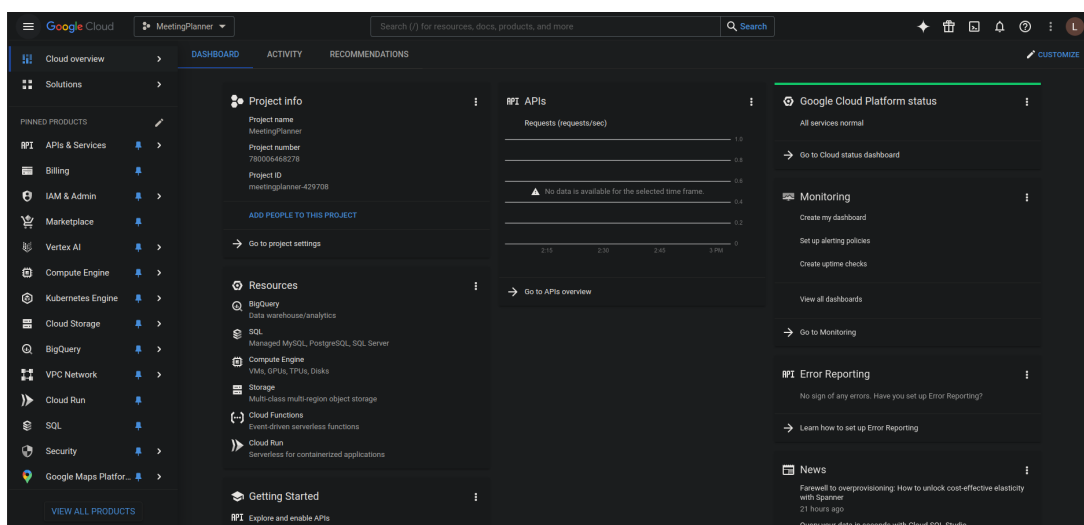
Prije izrade projekta trebaju se definirati funkcionalnosti projekta. Cilj aplikacije je dohvaćanje svih dostupnih termina koje korisnik ima u svome Google kalendaru te sinkronizacija u jedan ispis koji će organizatoru sastanka prikazati kada su dostupni svi planirani korisnici i odabrana dvorana u kojoj 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 kalendar (kalendar organizatora i kalendar korisnika): Nakon pronalaska slobodnog termina, sustav će omogućiti organizatoru upis sastanka u svoj kalendar kao i njihove kalendar.
- 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 interagirati pomoću HTTP poziva.

5.2. Google Workspace

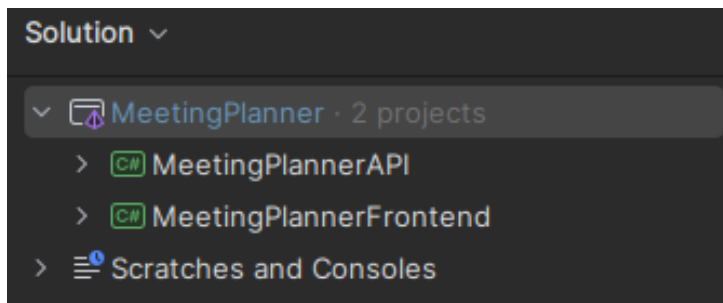
Google Workspace nam omogućava platformu za integraciju Google Workspace alata kao što su Maps, Calendar ili Sheets. Za ovaj projekt su nam potrebni Google Workspace APIs da integriramo kalendar s našim rješenjem. Da možemo koristiti te servise moramo prvo stvoriti projekt u Google Cloud konzoli te omogućiti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



Slika 2: Google Cloud console (Izvor: autor)

5.3. Stvaranje projekta

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



Slika 3: 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.

5.4. 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 konfigurirati autentifikaciju projekta u *Program.cs* datoteci.

```
builder.Services.AddAuthentication(options =>
{
    options.DefaultAuthenticateScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    options.DefaultSignInScheme = CookieAuthenticationDefaults.AuthenticationScheme;
    options.DefaultChallengeScheme = GoogleDefaults.AuthenticationScheme;
})
.AddCookie()
.AddGoogle(options =>
{
    var googleAuthNSection = builder.Configuration.GetSection("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 4: Postavljanje autentifikacije (Izvor: autor)

Kod konfiguriranja 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 5: 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 6: 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 7: Login servis (Izvor: autor)

5.5. 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 8: 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 9: 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 backend koji sadrži API endpointove. Zahtjevu se dodaje token u *AuthenticationHeaderValue* da se mogu dohvatiti podaci i šalje se zahtjev na definirani endpoint "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 10: Slanje zahtjeva na API endpoint 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 11: Metoda za upravljanjem povratnog JSON teksta (Izvor: autor)

Ovo je sva potrebna logika za prednji dio sustava engl. *Frontend*. Potrebno je implementirati pozadinski dio eng. *Backend*. Pozadinski 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")]
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 12: Kalendar "events" endpoint (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 13: Metoda za dohvaćanje instance Google kalendar servisa (Izvor: autor)

Drugi dio procesa je dohvaćanje *Events.List* resursa i mapiranje u obliku *SimpleEventModel*.

```

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();

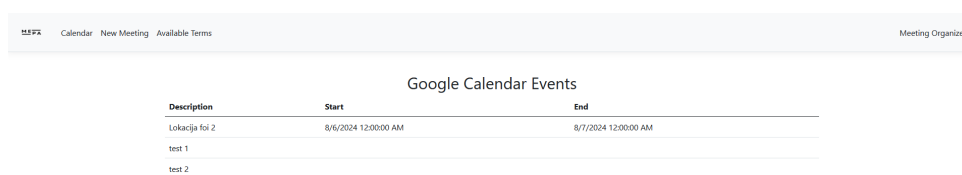
    return simpleEvents;
}

```

Slika 14: Metoda GetSimpleEventAsync (Izvor: autor)

Nakon implementacije metode naša funkcionalnost je potpuna, te se kalendar vraća u

obliku tablice prijavljenom korisniku.



Description	Start	End
Lokacija f0i 2	8/6/2024 12:00:00 AM	8/7/2024 12:00:00 AM
test 1		
test 2		

Slika 15: Prikaz kalendara na web stranici (Izvor: autor)

5.5.1. 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.

5.6. 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.

Check Free/Busy Times

Attendees (comma separated emails):

Event location:

Start Date:

End Date:

Start Date	End Date	Start Time	End Time
22/08/2024	23/08/2024	00:00	00:00

Slika 16: 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* endpoint, koji zatim proslijeđuje 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 => 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 => new TimePeriod
    {
        Start = (DateTime)b.Start,
        End = (DateTime)b.End
    }).ToList();

    return busyPeriods;
}
```

Slika 17: 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 18: 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;
}

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

    foreach (var freePeriod in freePeriods)
    {
        var currentFreeStart = 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 19: 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.

5.7. Unos sastanka

Kao završni dio aplikacije potrebno je omogućiti korisniku da unosi aktivnosti u kalendar korisnika.

6. Zaključak

.NET okruženje za skriptiranje u GNU/Linux naredbenom retku pruža nove mogućnosti za razvoj i automatizaciju. Kroz rad je demonstriran veći broj Bash i .NET skripti i demonstrirana je integracija .NET alata i Bash ljuške koristeći .NET alat dotnet-shell. Kroz primjere su prikazane prednosti i nedostatci oba sustava. .NET poboljšava interoperabilnost time što se skripte mogu direktno koristiti na svim operacijskim sustavima koji imaju instalirano .NET okruženje. Također, .NET pruža dodatne funkcionalnosti preko svojih biblioteka i NuGet paketa što omogućuje jednostavno razvijanje kompleksnijih skripti čime se još više mogu poboljšati radni procesi. Prednost Bash skripti je što rade na svim GNU/Linux distribucijama koje koriste Bash ljušku bez potrebe za dodatnim instalacijama i zahtjevaju manje resursa od .NET skripti čime su lakše za sustave. Da zaključim, sinergija .NET okruženja i Bash skriptnog jezika u GNU/Linux operacijskom sustavu omogućuje moćno okruženje za projekte automatizacije iz razloga što se može koristiti najbolje od oba jezika pri pisanju skripti, .NET elementi za kompleksne unaprijed pripremljene funkcionalnosti, a Bash za GNU/Linux specifične radnje ukoliko ima potrebe za njima.

Popis literature

- [1] M. Biehl, *API Architecture* (API-University Series). CreateSpace Independent Publishing Platform, 2015., ISBN: 9781508676645. adresa: <https://books.google.ba/books?id=6D64DwAAQBAJ>.
- [2] 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.)
- [3] 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.
- [4] IBM, *What is a REST API?* <https://www.ibm.com/topics/rest-apis> (Pristupano: 1.8.2024.).
- [5] Microsoft, *RESTful web API design*, <https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design> (Pristupano: 1.8.2024.), 2023.
- [6] C. Tyler, *ASP.NET MVC Tutorial for Beginners: What is, Architecture*, <https://example.com> (Pristupano: 12.8.2024.), 2024.
- [7] GeeksforGeeks, *MVC Design Pattern*, <https://www.geeksforgeeks.org/mvc-design-pattern/> (Pristupano: 12.8.2024.), veljača 2024.
- [8] 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: 12.8.2024.), 2022.
- [9] 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: 12.8.2024.), 2022.
- [10] 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: 12.8.2024.), 2023.

Popis slika

1.	Prikaz MVC u ASP.NET MVC projektu (Izvor: autor)	5
2.	Google Cloud console (Izvor: autor)	8
3.	Rješenje MeetingPlanner (Izvor: autor)	9
4.	Postavljanje autentifikacije (Izvor: autor)	10
5.	Metoda LoginWithGoogle (Izvor: autor)	10
6.	Metoda Callback (Izvor: autor)	11
7.	Login servis (Izvor: autor)	11
8.	Pregled kalendara (Izvor: autor)	12
9.	Pregled kalendara kontroler (Izvor: autor)	13
10.	Slanje zahtjeva na API endpoint za dohvaćanje kalendara (Izvor: autor)	13
11.	Metoda za upravljanjem povratnog JSON teksta (Izvor: autor)	14
12.	Kalendar "events" endpoint (Izvor: autor)	14
13.	Metoda za dohvaćanje instance Google kalendar servisa (Izvor: autor)	15
14.	Metoda GetSimpleEventAsync (Izvor: autor)	15
15.	Prikaz kalendara na web stranici (Izvor: autor)	16
16.	Web sučelje za prikaz dostupnih termina (Izvor: autor)	17
17.	Metoda <i>GetFreeBusyInformationAsync</i> (Izvor: autor)	17
18.	Dohvaćanje zauzetih termina odabrane lokacije (Izvor: autor)	18
19.	Metode za pretvaranje zauzetih termina u slobodne (Izvor: autor)	18