Vilniaus universitetas

ListPlaylist

Kompiuterių architektūros projektinio darbo aprašas

Informacinių sistemų inžinerija 1k.

Komandos nariai

Matas Remeškevičius
Vladas Zvonkus
Karolis Medekša
Paulius Mykolaitis
Adomas Naus
Benas Žitkevičius

Darbo vadovas

Marius Liutvinavičius

Vilniaus universitetas, Matematikos ir informatikos fakultetas

Vilnius, 2019

Turinys

Projektinio darbo tikslas, uždaviniai 4
Projektinio darbo tikslas 4
Projektinio darbo uždaviniai 4
Komandos narių pasiskirstymas kompetencijomis 5
Projekte naudojamos technologijos 6
Front-end 6
HTML 6
CSS 6
JavaScript 7
ReactJS 7
React Router 8
React Redux 8
Back-End 9
ASP.NET 9
C# programavimo kalba 9
ASP.NET Core 9
REST 10
Funkciniai reikalavimai – vartotojo panaudos diagramos 11
Aplikacijos veikimas – proceso diagrama 12
Jungtys tarp skirtingų aplikacijos dalių 12
Išorinės API paslaugos 14
Autentifikacija 14
Grojaraščių migracija 15
Autentifikacija ir autorizacija 18
OAuth 2.0 18
Aplikacijos publikavimas 20
Kūrimo metu kilę sunkumai 21

Projekto kūrimo žingsniai 22
Įrankiai, reikalingi projekto kūrimui 22
Projekto struktūra 22
Projekto kūrimo žingsniai 23
Aplikacijos paleidimas 26

Projektinio darbo tikslas, uždaviniai

Projektinio darbo tikslas

Sukurti pilnai funkcionuojančią *WEB* aplikaciją panaudojant *ASP.NET MVC Framework* technologiją. Aplikacija turi suteikti vartotojui galimybę prisijungti panaudojant Google bei Spotify paskyras ir atlikti grojaraščių migraciją iš vienos paskyros į kitą.

Projektinio darbo uždaviniai

- Sukurti *ASP.NET MVC* aplikaciją.
- Sukurti aplikacijos *Front-end* dalies dizainą bei pritaikyti jį projekte.
- Įdiegti vartotojų autorizacijos bei grojaraščių migracijos logiką.
- Ištestuoti aplikacijos veikimą.
- Publikuoti aplikaciją.
- Sukurti projekto dokumentacijos aprašą.

Komandos narių pasiskirstymas kompetencijomis

Adomas Naus

Front-end programavimas, puslapio dizaino kūrimas.

Karolis Medekša

Darbo organizavimas, *Front-end* programavimas, puslapio dizaino kūrimas, dokumentacija.

Vladas Zvonkus

Front-end programavimas, *Back-end* programavimas.

Matas Remeškevičius

Back-end programavimas, aplikacijos testavimas, publikavimas.

Paulius Mykolaitis

Dokumentacija.

Benas Žitkevičius

Dokumentacija.

Projekte naudojamos technologijos

Front-end

HTML

Tai kompiuterinė žymėjimo kalba, naudojama pateikti turinį internete. HTML yra bet kurios interneto svetainės kūrimo proceso pagrindas, be kurio interneto puslapis neegzistuotų. HTML kodas užtikrina teksto ir paveiksliukų tinkamą formatą, kad interneto naršyklėje galėtume matyti, kaip viskas turėtų atrodyti pagal parašytą HTML kodą. Be HTML kodo, interneto naršyklė nežinotų, kaip pavaizduoti tekstą ar paveikslėlius ar kitus elementus. HTML taip pat aprūpina interneto puslapį su paprastomis struktūromis, kurių dizainas gali būti pakeistas CSS pagalba. Paprastas pavyzdys : HTML – stuburas, o CSS – oda.

Šią kompiuterinė žymėjimo kalbą naudosime internetinio puslapio dizaino kūrimui, nuorodų kūrimui bei teksto formavimui. Tai bus mūsų internetinio puslapio stuburas.

https://www.youtube.com/watch?v=zP6iJbpPcIw - įvadinis filmukas į HTML. Paaiškina kas tai, kaip veikia, kokios yra HTML funkcijos.

CSS

Kalba, skirta nusakyti kita struktūrine kalba aprašyto dokumento vaizdavimą. Nors ankstesnėse *HTML* versijose buvo nemažai išvaizdą aprašančių elementų, bet rekomenduojama *HTML* kalba žymėti dokumento sandarą, o išvaizdą (teksto spalvas ir pan.) aprašyti atskirame *CSS* dokumente.

- Toks dokumentas užima mažiau vietos ir greičiau pasirodo vartotojo naršyklėje (įvairių puslapių išvaizdą aprašantis *CSS* dokumentas iš serverio atsisiunčiamas tik vienąkart);
- Toks dokumentas lengviau bei kokybiškiau apdorojamas automatiškai, todėl tokius dokumentus geriau indeksuoja paieškos sistemos;
- Naudojant *CSS*, lengviau iškart keisti visų puslapių išvaizdą;
- Taip paprasčiau pasiekti, jog šiuos puslapius įvairios naršyklės rodytų vienodai;
- Esant ribotam perdavimo kanalui galima siųsti tik patį *HTML* dokumentą, bei nesiųsti jo *CSS;*
- Galima siųsti tik patį *HTML* dokumentą, bei nesiųsti jo *CSS* jei naršyklė nepajėgi jį atvaizduoti;

- Panaudojus kelis skirtingus *CSS* dokumentus, tą patį *HTML* dokumentą galima gerai pritaikyti skirtingoms naršyklėms, pvz., vienaip atvaizduoti kompiuterio ekrane, o kitaip – mobiliajame telefone.

https://www.youtube.com/watch?v=0afZj1G0BIE - vaizdo įrašas, kuris parodo pačiu paprasčiausiu būdu *CSS* veikimo principą ir jo funkcijas - braižant. Taip pat parodoma, kaip yra rašomas *CSS* kodas.

JavaScript

Viena populiariausių programavimo kalbų, dažniausiai naudojama internetinių puslapių kūrimui. Paprastai JavaScript kalbos kodas įtraukiamas į *HTML* puslapius, tokiu būdu išplečiant *HTML* kalba kurtus puslapius naujomis funkcijomis – galimas anketų parametrų tikrinimas, naujų langų atidarymas, suskleidžiamos hierarchinės struktūros rodymas, išsiskleidžiantis meniu ir daug kitų interaktyvumo formų. Apibendrinant, *JavaScript* pagalba puslapis gali pilnai veikti su visomis tipinio internetinio puslapio funkcijomis.

https://guide.freecodecamp.org/javascript/ - įvadas į *JavaScript*. Aprašyta informacija, kuri gali būti reikalinga internetinio puslapio kūrimui.

ReactJS

Tai *JavaScript* biblioteka, skirta *UI* komponentų kūrimui. *Facebook* sukūrė *ReactJS* tam, kad galėtų lengviau valdyti *UI* elementų išdėstymą. Ši biblioteka taip pat palengvina sudėtingos vartotojo sąsajos (*UI*) kūrimą. *UI* yra grafinis programos, arba šiuo atveju internetinio puslapio, išdėstymas. Vartotojo sąsaja susideda iš mygtukų, ant kurių gali paspausti vartotojas, tekstas, kurį jis skaito, paveikslėliai, teksto įvedimo laukeliai ir visi kiti objektai, su kuriais vartotojas sąveikauja. Tam ir yra reikalingas *ReactJS* - būtent *UI* kūrimui. React galima rašyti standartiniais *React* moduliais/objektais arba *JSX* kalba, kuri leidžia tiesiogiai įterpti *HTML* į *JavaScript*. Tai leidžia glaudžiai susijusią atvaizdavimo logiką ir žymes (*markup*) laikyti vienoje vietoje, išnaudoti pilnas *JavaScript* galimybes ir funkcijas. Tai nėra karkasas, jis nesuteikia papildomų funkcijų bendravimui su serveriu, navigacijai aplikacijoje, duomenų valdymui ir panašiai.

https://www.tutorialspoint.com/reactjs/index.htm - įvadinė pamoka i *ReactJS*, kurioje paaiškinama bibliotekos pagrindinės funkcijos bei veikimo principas.

React Router

Tai yra *React* biblioteka, skirta maršruto nustatymui, kuri leidžia naršyti po programą ar puslapį nuorodų pagalba. Maršrutas susieja programą su naršyklės siūlomomis naršymo funkcijomis: adresų juosta ir naršymo mygtukais. *React Router* padeda parašyti kodą, rodantį tam tikrus programos komponentus tik tuo atveju, kai maršrutas atitinka mūsų apibrėžtą.

https://www.freecodecamp.org/news/beginner-s-guide-to-react-router-53094349669/ - įvadas į *React Router*.

React Redux

Tai yra *React* biblioteka, kuri laiko atskirų React komponentų būsenos informaciją vienoje vietoje. Tai lyg visuotinis objektas, kuriame bus kaupiama informacija, kuri bus naudojama įvairiems tikslams vėliau programoje (pvz., priimant sprendimus, kuriuos komponentus atiduoti ir kada, pateikiant saugomus duomenis ir pan.).

https://medium.com/javascript-in-plain-english/the-only-introduction-to-redux-and-react-redux-youll-ever-need-8ce5da9e53c6 - nuosekli *React Redux* pamoka. Paaiškina, kaip biblioteka veikia, kodo rašymo ypatumus.

Back-End

ASP.NET

ASP.NET yra atviro kodo karkasas, skirtas programinės įrangos kūrimui. *.NET* architektūra yra sudaryta iš komponentų:

- Kalba – pagrindinės kalbos, naudojamos *.NET* architektūroje yra *C#*, kurią mes ir naudojame, *F#* ir *Visual Basic*. *ASP.NET* yra parašyta su *CLR* (*Common Language Runtime*), kas leidžia rašyti *ASP.NET* kodą naudojant bet kokią pasirinktą *.NET* kalbą.
- Biblioteka - *.NET* karkasas turi keletą bibliotekų. Dažniausiai interneto svetainėms kurti naudojama biblioteka yra *Web* biblioteka. Ji turi visus reikalingus komponentus, naudojamus sukurti *.NET* internetinėms aplikacijoms.

https://www.guru99.com/what-is-asp-dot-net.html - aprašo kas yra *ASP.NET* bei iš ko tai sudaryta.

C# programavimo kalba

C# yra programavimo kalba, turinti panašią struktūrą kaip *Java* ar *C* bei naudojanti *.NET* karkasą. Su ja kuriamos interneto svetainės, *Windows* ir *Android* programos, *Unity* naudojantys žaidimai ir kitos programos.

https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework - pamini kas yra *C#* kalba ir aprašo kaip ji veikia.

https://csharp-station.com/what-is-c-used-for/ - aprašo kam naudojama *C#* kalba bei kuo ji panaši į *C++*.

ASP.NET Core

Galinėje aplikacijos dalyje (*angl. Back end*) veikianti technologija – ***ASP.NET Core*** karkasas. *ASP.NET Core* karkasas priklauso įmonės Microsoft sukurtai ***ASP.NET*** technologijų šeimai. *ASP.NET* Core galima vertinti kaip atnaujintą plačiai naudojamo *ASP.NET MVC* karkaso versiją. Palyginus su pastaraja technologija, *.NET Core* turi privalumų, naudingų šiam projektui:

- Suderinamumas tarp platformų: *ASP.NET Core* veikia tiek ant ***Windows***, tiek ant ***Linux*** ar ***Mac*** serverių. Naudojant šią technologiją kils mažiau problemų perkeliant aplikaciją iš vieno serverio į kitą.
- Galima kurti internetinę aplikaciją ir API sąsają viename projekte, naudojant *ASP.NET MVC* šis procesas būtų sudėtingesnis.

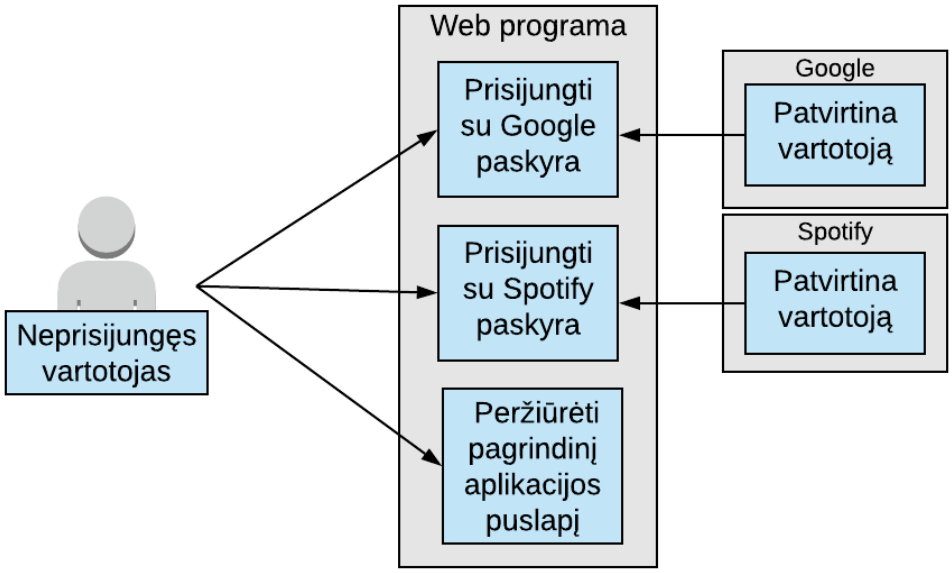
REST

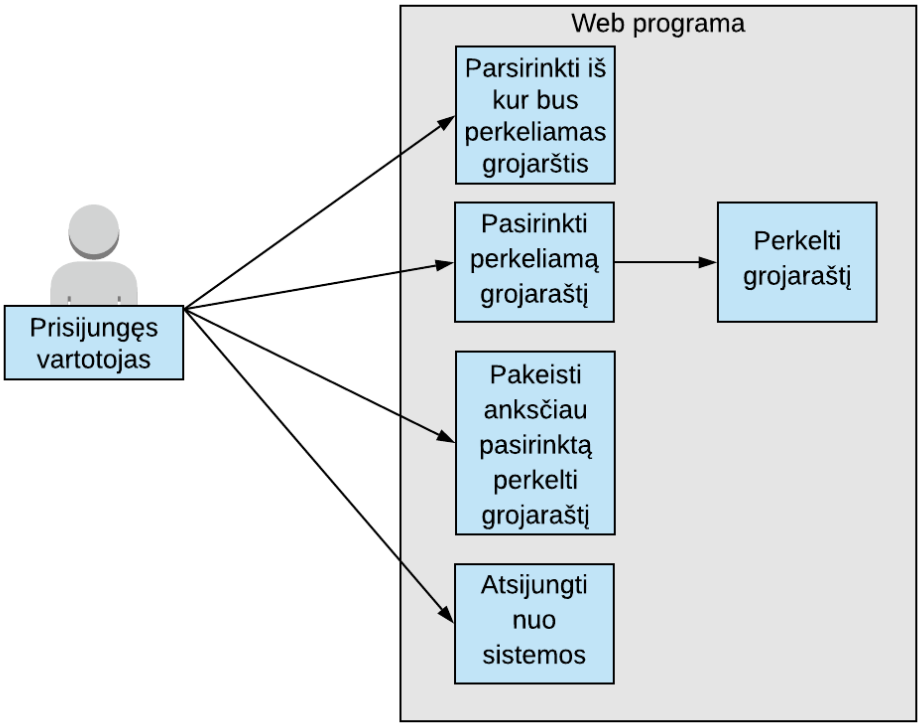
REST (*Representational State Transfer*) architektūra apibrėžia kokio tipo komandos bus siunčiamos tarp serverio ir kliento.

- *REST* atskiria serverio programinę įrangą nuo kliento. Tai leidžia bet kada keisti kliento kodą nepaveikiant serverio ir atvirkščiai.
- Sistemos, naudojančios *REST*, yra „*Stateless*“. Tai reiškia, jog nei serveris, nei klientas neturi žinoti ką kitas veikia norint suprasti vienas kitam siunčiamas komandas.

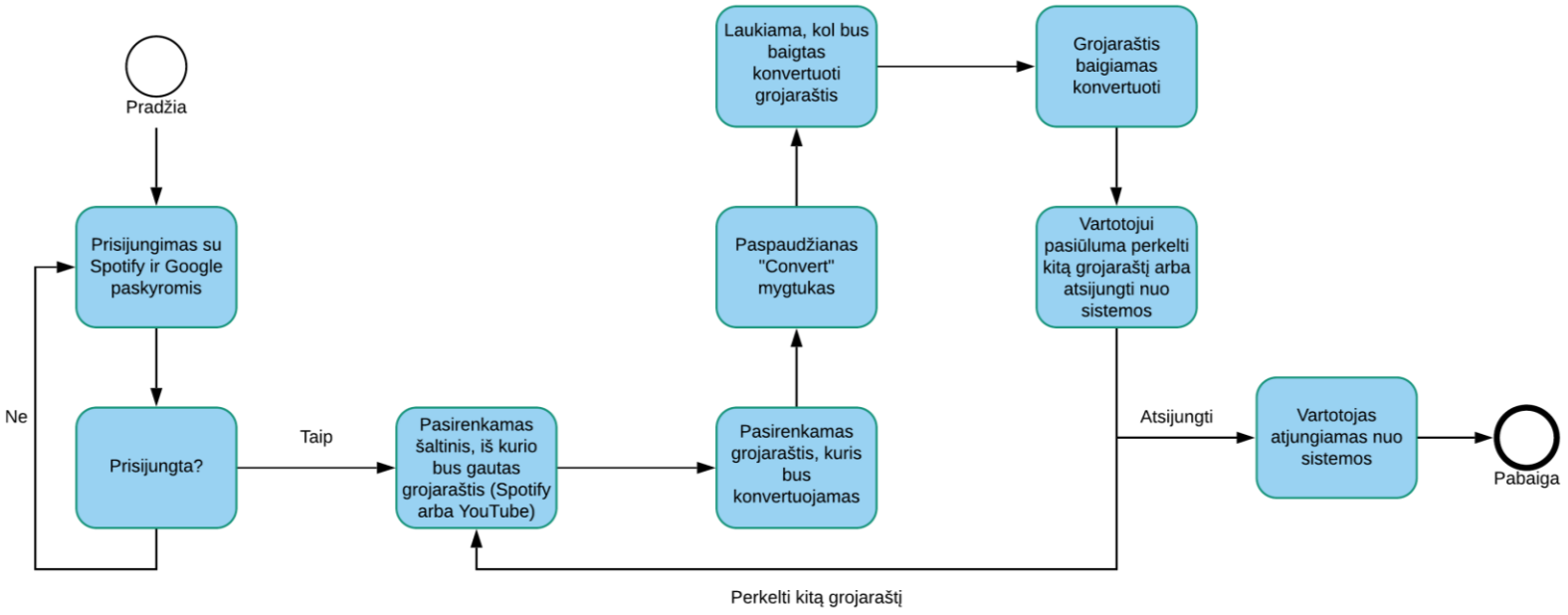
https://www.codecademy.com/articles/what-is-rest - aprašo pagrindinius *REST* architektūros veikimo principus.

Funkciniai reikalavimai – vartotojo panaudos diagramos

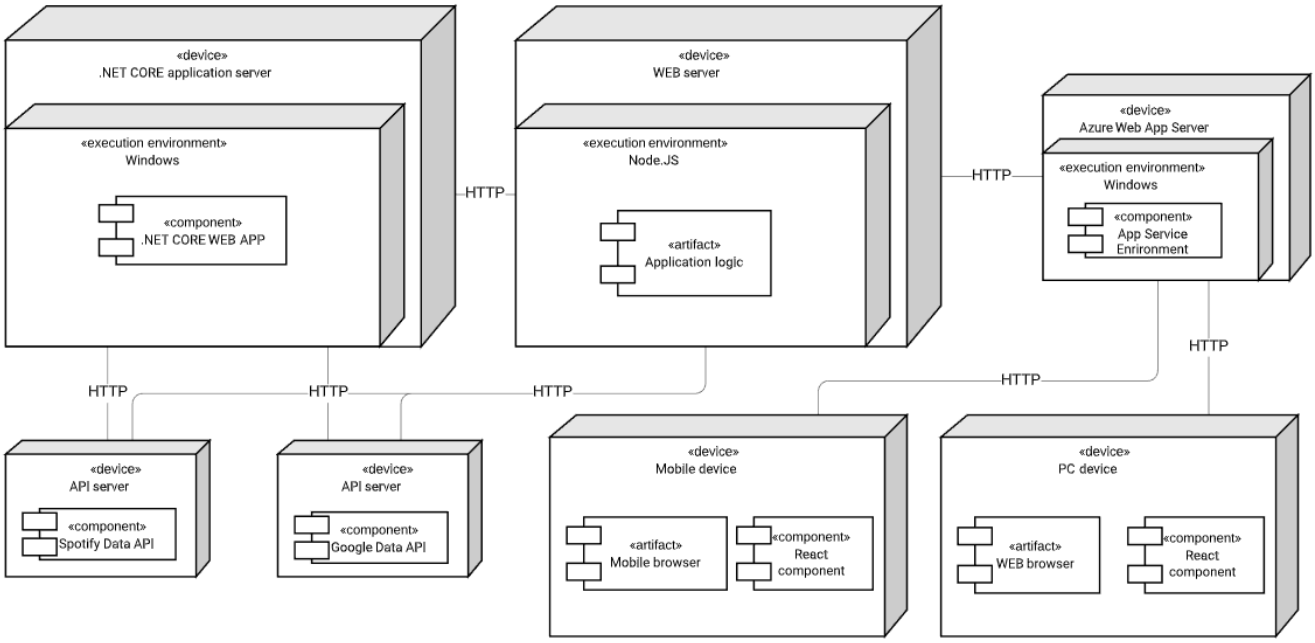




Aplikacijos veikimas – proceso diagrama



Jungtys tarp skirtingų aplikacijos dalių



Šios aplikacijos veikimas susidaro iš 3 pagrindinių dalių – tai **WEB serveris**, kuriame slypi visa puslapio išvaizda, autentifikacijos metodai bei dizaino bendravimas su aplikacijos logika, **.NET CORE application server** – *HTTP POST/GET* ryšiais bendravimas su *Spotify bei Google Data API*, grojaraščių kūrimo, dainų ieškojimo bei jų įterpimas į sukurtą grojaraštį logika, bei **Azure Web App Server –** tai *Azure Web Service*, kuriame patalpinta mūsų *WEB* aplikacija, jos logika bei sukuriamas išorinis priėjimas prie *WEB* aplikacijos.

WEB serveryje slypi *React* komponentai, kurie suteikia vartotojui išvaizdų dizainą, perteikia visą aplikacijos logiką grafiškai bei leidžia fiziškai naudotis ja.

.NET CORE application server slypi visa aplikacijos logika – kuriamas naujas grojaraštis vartotojo pasirinktoje aplinkoje, ieškomos dainos iš grojaraščio, surasti rezultatai įkeliama į naująjį grojaraštį. Ieškojimas, sukūrimas bei įkėlimas vyksta *HTTP POST/GET* metodais aprašytais aplikacijos valdytojuose (angl. *controllers*).

Azure Web App Server tai mokama paslauga, kuri leidžia įkelti mūsų sukurtą *WEB* aplikaciją, suteikti jai išorinį priėjimą, stebėti puslapio resursus. Ši paslauga paskiria vieną procesorių bei 3,5GB RAM, taip pat 1GB talpos įkelti aplikaciją.

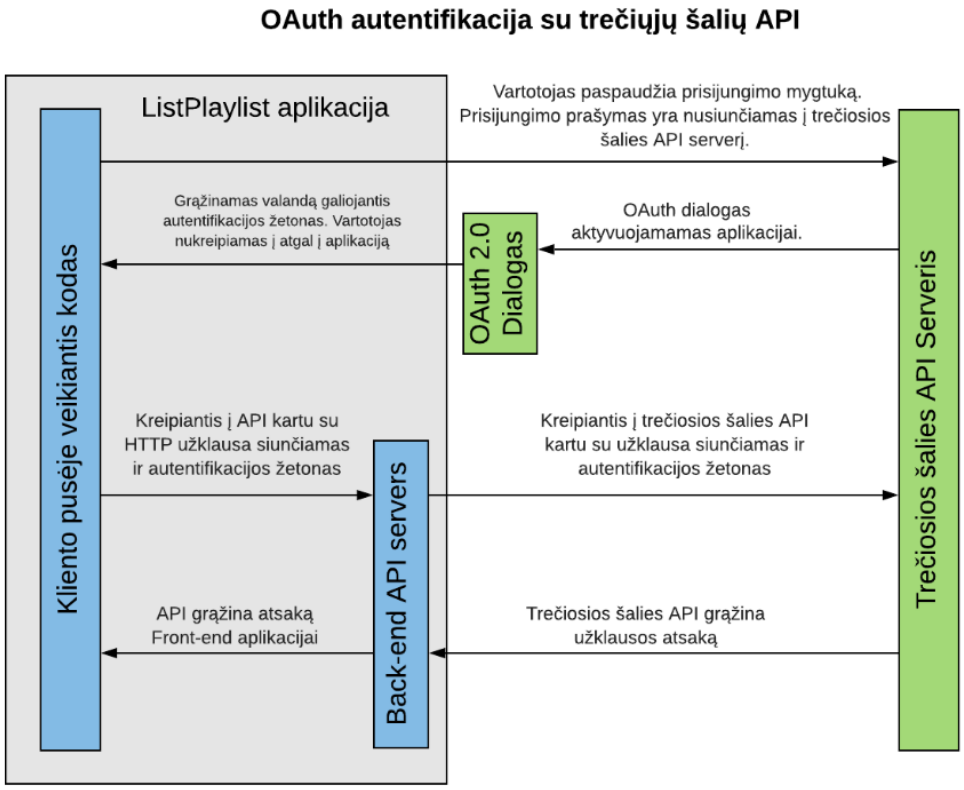
Išorinės API paslaugos

Trečiųjų šalių *API*(*YouTube Data v3* ir *Spotify API*) yra naudojami trims svarbiausioms užduotims atlikti:

- Vartotojų autentifikacijai
- Informacijos apie vartotojo turimus grojaraščius gauti
- Perkelti grojaraščius iš vienos sistemos į kitą

Autentifikacija

Autentifikacija tiek su *Spotify*, tiek su *Google* vykdoma identiškai. Naudojamas *OAuth 2.0* autentifikacijos metodas. Vartotojui paspaudus prisijungimo mygtuką atskirame lange paprašoma prisijungti su pasirinktos sistemos prisijungimo duomenimis. Prisijungus vartotojas grąžinamas atgal į aplikaciją, o aplikacijai suteikiama informacija apie prisijungusį vartotoją bei laikinas, valandą galiojantis autentifikacijos žetonas. Žetoną galima iškeisti į ilgiau galiojantį *Access* žetoną bei *Refresh* žetoną. Tiesa, kadangi vartotojams nebūtina būti prisijungus prie sistemos ilgą laiką, užklausoms autorizuoti naudojamas laikinas autentifikacijos žetonas.



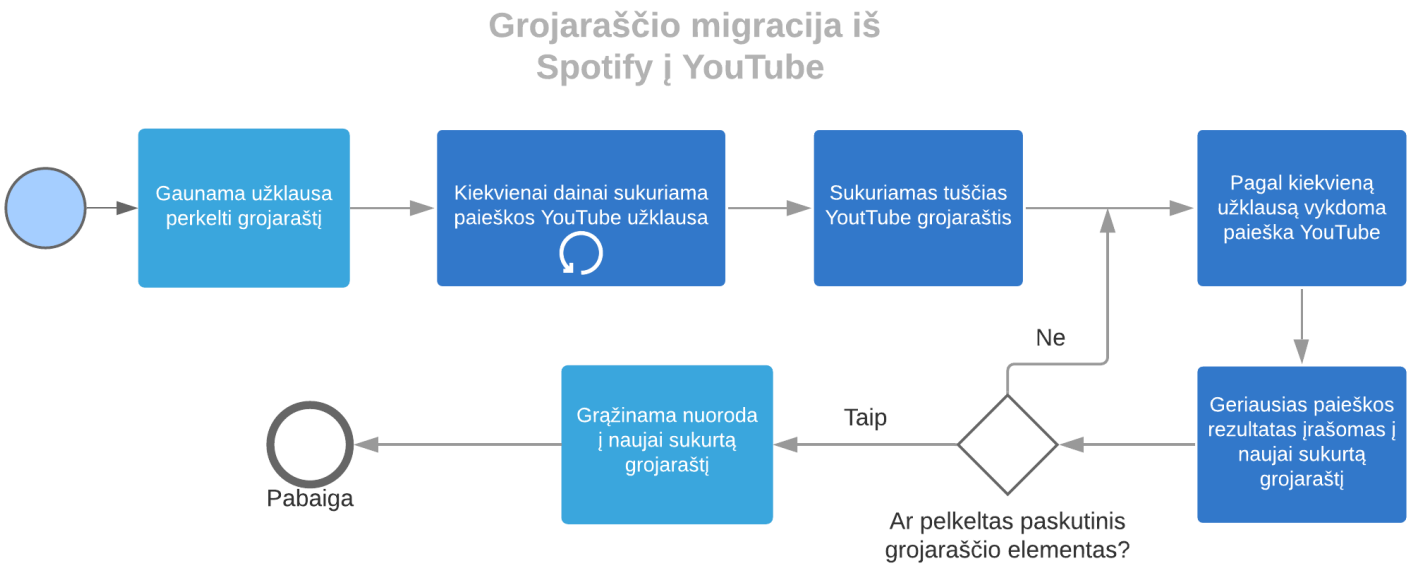
Grojaraščių migracija

Google bei *Spotify API* veikia skirtingai, tad perkeliant grojaraščius iš *YouTube* į *Spotify* ir iš *Spotify* į *YouTube* reikalingi skirtingi algoritmai.

Perkeliant grojaraštį iš *Spotify* į *YouTube* galinėje aplikacijos dalyje atliekama tokia veiksmų seka:

- Gaunama visų grojaraštyje esančių dainų informacija: atlikėjas ir dainos pavadinimas.
- Pagal kiekvienos dainos duomenis sukuriamos užklausos vaizdo įrašų paieškai *YouTube*.
- Sukuriamas tuščias *YouTube* grojaraštis.

- Kiekvienai dainai vykdoma vaizdo įrašo paieška, o geriausias rezultatas išsaugomas naujai sukurtame grojaraštyje.

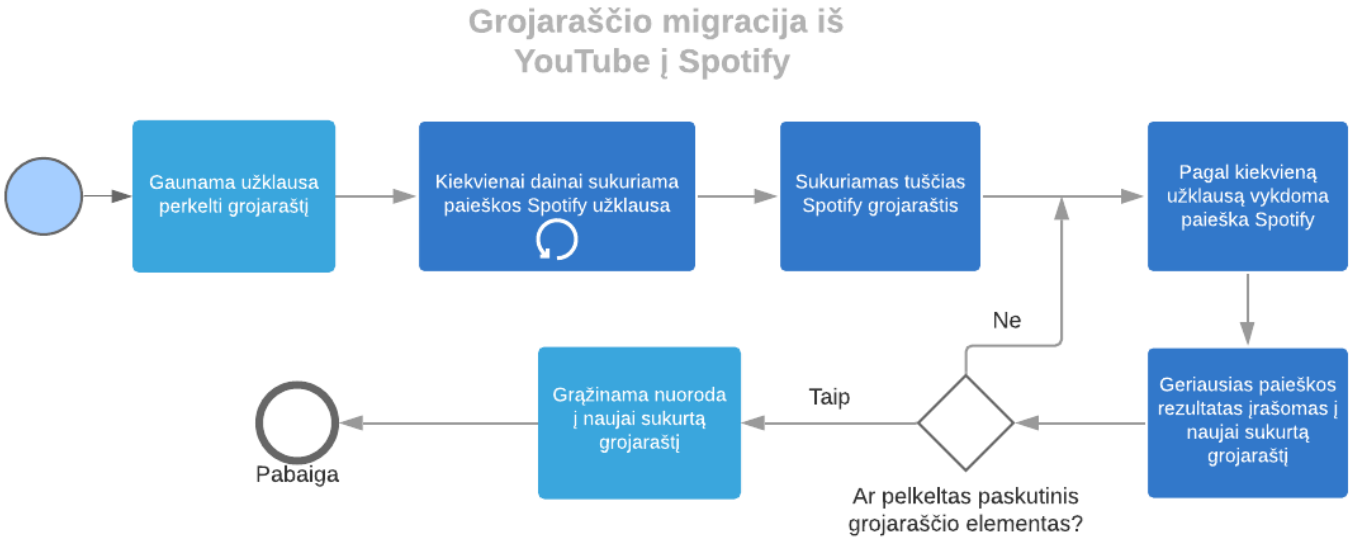


Perkeliant grojaraštį į *Spotify* atliekama tokia veiksmų seka:

- Gaunami visi grojaraščio vaizdo įrašų pavadinimai.
- Iš kiekvieno pavadinimo naudojant specialią funkciją sukuriamas paieškos *Spotify* tekstas:

```
public string FormatTitle(string title)
{
    title = title.ToLower();

    int found1 = title.IndexOf("(");
    if (found1 != -1)
        title = title.Substring(0, found1 - 1);

    int found2 = title.IndexOf("[");
    if (found2 != -1)
        title = title.Substring(0, found2 - 1);

    string[] trimWords =
    {
        "feat", "ft.",
        "official music video", "music video", "official video", "video",
        "with lyrics", "lyrics",
        "live performance", "live", "vevo",
        "/", "\\", "&", "!", "?", "@", "#", "-", "\"", ":",
    };

    foreach (string trimWord in trimWords)
        title = title.Replace(trimWord, "");

    return title;
}
```

- Sukuriamas tuščias *Spotify* grojaraštis.
- Kiekvienam vaizdo įrašui vykdoma paieška *Spotify* sistemoje, jei jokios sutampančios dainos nerandama, į grojaraštį neįrašoma nieko.



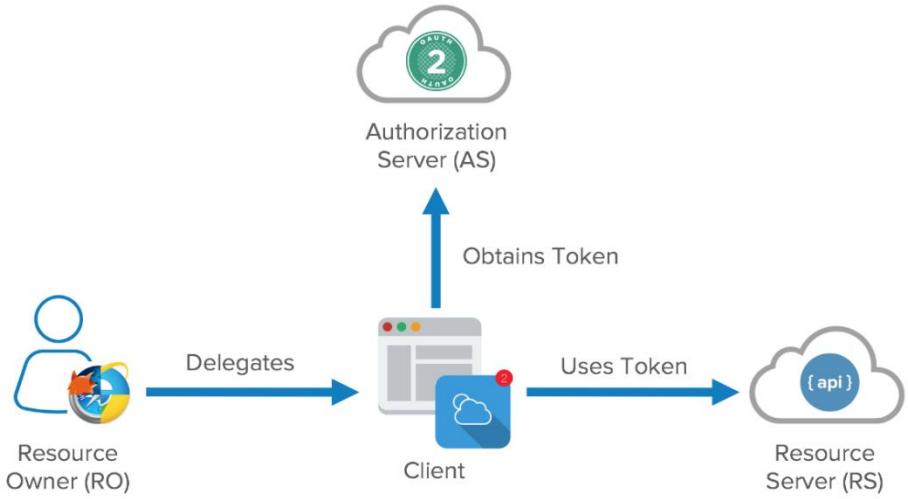
Autentifikacija ir autorizacija

OAuth 2.0

Atviras autentifikavimo protokolas (schema), leidžiantis suteikti trečiajai šaliai ribotą prieigą prie vartotojo saugomų išteklių, nesuteikiant jai (trečiajai šaliai) vartotojo vardo ir slaptažodžio. Tiksliau tariant, „OAuth“ yra standartas, kurį programos gali naudoti tiekdamos kliento programoms „saugią deleguotą prieigą“.

Kaip tai veikia?

„*OAuth*“ veikia per *HTTPS* ir autorizuoja įrenginius, *API*, serverius ir programas su vienkartiniais prieigos raktais, o ne prisijungimo duomenimis. Pagrindinė autentifikacija vis dar naudojama kaip primityvi API autentifikavimo forma serverio pusėje esančioms programoms: užuot siųsdamas vartotojo vardą ir slaptažodį serveriui su kiekviena užklausa, vartotojas siunčia laikiną API rakto ID. Laikino rakto pagalba, vartotojas gali susieti vieną servisą su kitu (Spotify su YouTube ir atvirkščiai).



Kodėl naudosime?

Norint sukurti geresnę WEB sistemą, buvo sukurtas vienkartinis prisijungimas (SSO). SSO prisijungimo principas - perduodamas vienkartinis autentifikavimo žetonas (token), kurio pagalba vartotojas gali prisijungti prie atitinkamos sistemos su mažiau pastangų. Kai vartotojas bus autentifikuotas, su mažai pastangų galės sėkmingai susieti Spotify su YouTube ar atvirkščiai, to pasėkoje, galėsime konvertuoti grojaraščius. Tai puikiai padės pasiekti mūsų projekto tikslą - su

vienkartiniu prisijungimu prie vienos iš šių platformų konvertuoti Spotify grojaraštį į YouTube grojaraštį arba atvirkščiai.

- Trumpai tariant, OAuth 2.0 veikimas:
- Programa reikalauja vartotojo leidimo prisijungiant;
- Vartotojas prisijungia prie programos ir pateikia įrodymus, kad tai būtent jis;
- Programa pateikia serverio autorizacijos liudijimą, kad gautų raktą, kurį vėliau panaudos autentifikacijai;
- Raktas yra apribotas, kad būtų galima pasiekti tik tai, ką vartotojas įgaliojo konkrečiai programai.

Aplikacijos publikavimas

Aplikacija publikuojama per *Azure Web Services*. Ši paslauga palaiko tiek *Windows*, tiek *Linux* platformas, leidžia nuolatos atnaujinti programinį kodą su *Git, GitHub* ar *DevOps*.

Iš pradžių yra sukuriamas *Web Server Resource*, kuriame aprašome savo aplikaciją, nustatome jai tinkamą aplinką ir sukuriame paleidimo failą. Sukurtas paleidimo failas tuomet yra įkeliamas į mūsų projektą, nustatomi tinkami maršrutai tarp aplikacijos komponentų bei išorinių *API* ir programa yra įdiegiama į *Azure* debesį. *Azure* leidžia užtikrinti 24/7 puslapio veikimą, pilną palaikymą bei pranešimą apie įvairiausias klaidas, iškylančias aplikacijoje.

Kadangi aplikacija yra glaudžiai susijusi su *GitHub*, Azure užtikrina nuolatos atnaujinamą programą, kadangi paimama kaskart naujausia projekto versija.

Kūrimo metu kilę sunkumai

- Netinkamai pasirinktos technologijos.

  Projekto kūrimo pradžioje galinei aplikacijos daliai pasirinkta buvo ne *.NET Core*, o *.NET MVC* technologija. Neilgai trukus supratome, jog *MVC* karkasas nėra pati geriausia technologija poruoti su atskira *front-end ReactJS* aplikacija. Todėl viduryje projekto kūrimo teko perrašyti visą back-end kodą *.NET Core* karkasui.

  Taip pat projekto kūrimo pradžioje buvo išsikeltas tikslas integruoti į projektą ir duomenų bazę, daug dėmesio buvo skirta šio komponento projektavimui. Tačiau pastebėjus, jog duomenų bazė nebuvo reikalinga nei vartotojų autentifikacijai, nei grojaraščių migracijai, duomenų bazės buvo nuspręsta atsisakyti. Tam taip pat įtakos turėjo ir trečiųjų šalių *API* paslaugų naudojimo sąlygos, draudžiančios turinio saugojimą.

  Ištrauka iš *Spotify API* naudojimo sąlygų:

  b. **Local caching**. Except as set out in this paragraph, you will not locally cache any Spotify Content. Only when strictly necessary to enhance the performance of your SDA and its functionality, your SDA may locally cache (i) metadata and cover art or (ii) Conditional Downloads of sound recordings. Caching of Conditional Downloads of sound recordings under clause (ii) shall only be available to subscribers to the Premium Service. "**Conditional Downloads**" means time-limited offline syncing that is available to subscribers to our Premium Service.

  b. **Local caching**. Except as set out in this paragraph, you will not locally cache any Spotify Content. Only when strictly necessary to enhance the performance of your SDA and its functionality, your SDA may locally cache (i) metadata and cover art or (ii) Conditional Downloads of sound recordings. Caching of Conditional Downloads of sound recordings under clause (ii) shall only be available to subscribers to the Premium Service. "**Conditional Downloads**" means time-limited offline syncing that is available to subscribers to our Premium Service.

- Ydingas laiko planavimas.

  Kai kuriems projekto uždaviniams, planuojant projektą, nebuvo skirta pakankamai laiko, dėl ko kilo sunkumų stengiantis tilpti į turimo laiko rėžius kuriant projektą.

- *YouTube Data API* apribojimai.

  Pradėjus projekto kūrimą nebuvo atsižvelgta į tai, kad *Google* neseniai ženkliai sumažino *YouTube Data API* naudojimo kvotas (nuo 1000000 iki 10000 taškų). Grojaraščio sukūrimas ir atnaujinimas yra verti po 53 taškų. Tai reiškia, kad perkeliant iš *Spotify* į *YouTube* 100 dainų grojaraštį yra išnaudojama pusė paros kvotos.

Projekto kūrimo žingsniai

Šiame skyrelyje pateikta informacija turėtų padėti norintiems sukurti panašų projektą. Pateikiami svarbiausi projekto kūrimo žingsniai bei nurodomos nuorodos į naudingus resursus bei šio projekto failus.

Įrankiai, reikalingi projekto kūrimui

- *Node.js* kūrimo serveris bei *npm*.

  https://nodejs.org/en/

- *Git* klientas bei, galimai, grafinė *git* sąsaja.

  https://git-scm.com/downloads *git* klientas.

  https://www.gitkraken.com/ rekomenduojama grafinė *git* sąsaja.

- *.NET* programavimo įrankiai – *Visual Studio 2019 IDE*, *WEB development* įrankiai.

  https://visualstudio.microsoft.com/vs/ *Visual Studio 2019 IDE*

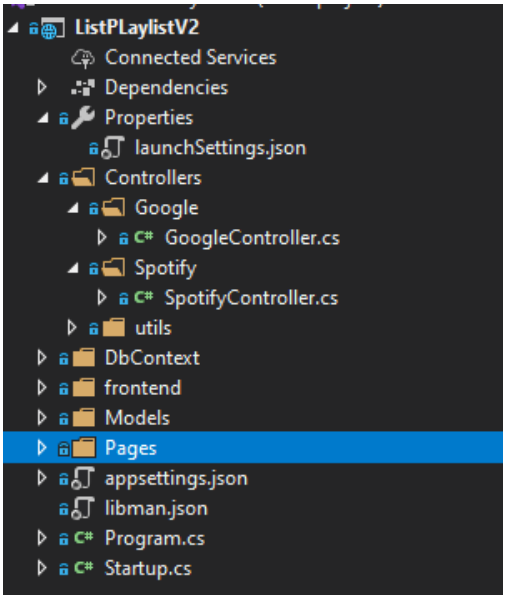
- *IDE* arba teksto redaktorius *ReactJS*, *HTML*, *CSS* redaguoti, pavyzdžiui:

  https://visualstudio.microsoft.com/vs/ *Visual Studio 2019 IDE*

  https://www.jetbrains.com/idea/ *IntelliJ IDEA*

  https://code.visualstudio.com/ *Visual Studio Code*

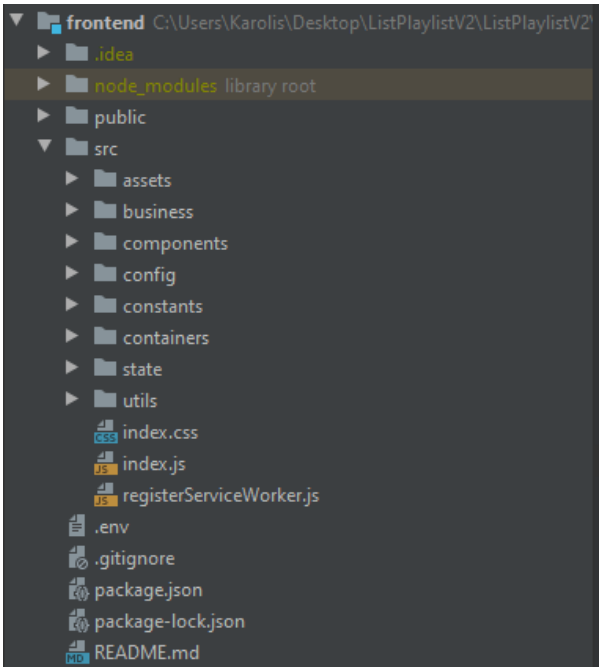
Projekto struktūra



Controllers – aplankas, su sukurto *API* galiniais taškais (angl. *endpoints*).

Models – aplankas su duomenų apdorojimo logika.

frontend – front-end aplikacijos dalies aplankas.



public – aplankas su bendrais puslapio failais.

src – aplankas su *front-end* kodu.

src/assets – šrifto ir nuotraukų failai.

src/business bei *src/utils* aplankuose saugomi autentifikacijos logikai reikalingi failai.

src/components ir *src/containers* aplankuose saugomi projekto *React* komponentai.

src/config – konfigūracinė projekto informacija.

src/constants – projekto konstantos.

.env – globalių konstantų failas.

src/state – *React Redux* būsenos failų konfigūracija.

Projekto kūrimo žingsniai

1. Susipažinkite su .NET Core bei ReactJS technologijomis.

   Naudingos nuorodos:

   https://docs.microsoft.com/en-us/dotnet/core/

   https://www.youtube.com/watch?v=--lYHxrsLsc

   https://reactjs.org/tutorial/tutorial.html

   https://www.freecodecamp.org/learn/front-end-libraries/react/

2. Sukurkite ASP.NET Core WEB API projektą. Sukurkite ReactJS projektą, naudodami create-react-app komandą.

   Naudingos nuorodos:

   https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api?view=aspnetcore-3.1&tabs=visual-studio

   https://reactjs.org/docs/create-a-new-react-app.html

3. Įdiekite šias *front-end* bibliotekas naudodami *npm install* komandą:

   react-redux, react-router-dom, react-router-redux, redux, redux-persist, redux-thunk.

Šios bibliotekos būtinos tinkamam *React-Redux* aplikacijos kūrimui. Norint pilnai atkartoti šį projektą gali reikėti įdiegti papildomas bibliotekas, apie tai turėtų informuoti teksto tvarkyklė.

4. Sukurkite svarbiausius aplikacijos puslapius bei jų dizainą. Pasistenkite, kad dizainas būtų pritaikytas mobiliesiems įrenginiams.

   Kaip pavyzdį galite naudoti *index.css, src/components* bei *src/containers* aplankuose esančius *.js* bei *.css* failus.

   Naudingos nuorodos:

   https://www.freecodecamp.org/learn/responsive-web-design/basic-html-and-html5/

   https://www.w3schools.com/html/html_responsive.asp

5. Sukurkite ir sukonfigūruokite *Redux store*. Prijunkite prie jos atitinkamus *React* elementus.

   Kaip pavyzdį galine naudoti *index.js* bei *src/state* esančius *.js* failus.

   Naudingos nuorodos:

   https://www.freecodecamp.org/learn/front-end-libraries/redux/

   https://www.freecodecamp.org/learn/front-end-libraries/react-and-redux/

   https://react-redux.js.org/introduction/quick-start

6. Sukurkite ir sukonfigūruokite projektus *Google* bei *Spotify* konsolėse.

   Naudingos nuorodos:

   https://cloud.google.com/resource-manager/docs/creating-managing-projects

   https://developer.spotify.com/dashboard

   https://levelup.gitconnected.com/how-to-build-a-spotify-player-with-react-in-15-minutes-7e01991bc4b6

7. Įdiekite vartotojo autorizaciją su *Google* ir *Spotify*.

   Tam patogu naudoti biblioteką *react-google-login*.

   Kaip pavyzdį galite naudoti *src/containers/Home/Home.js, src/business/authService.js* failus.

   Naudingos nuorodos:

https://developer.spotify.com/documentation/web-api/quick-start/

https://www.npmjs.com/package/react-google-login

https://levelup.gitconnected.com/how-to-build-a-spotify-player-with-react-in-15-minutes-7e01991bc4b6

8. Į *back-end* projektą įdiekite ir sukonfigūruokite bibliotekas:

   SpotifyAPI.Web, Google.Apis.Auth, Google.Apis.YouTube.v3

9. Sukurkite atitinkamas valdiklių klases bei *API endpoint'us* informacijai apie grojaraščius iš *API* gauti bei vykdyti grojaraščių migraciją.

   Kaip pavyzdį galite naudoti *Controllers/Google/GoogleController.cs*, *Controllers/Spotify/SpotifyController.cs* klases.

   Naudingos nuorodos:

   https://johnnycrazy.github.io/SpotifyAPI-NET/#about

   https://developers.google.com/youtube/v3/docs/?apix=true

   https://github.com/youtube/api-samples/tree/master/dotnet

10. Sujunkite aplikacijos *API* dalį su *front-end* dalimi.
11. Ištestuokite aplikaciją, raskite ir ištaisykite klaidas.
12. Publikuokite aplikaciją.

Aplikacijos paleidimas

1. Atsisiųskite aplikaciją iš https://github.com/MKarolis/ListPlaylistV2
2. Sukurkite ir sukonfigūruokite projektus *Google Developer Console* bei *Spotify*.
3. Per terminalą paleiskite komandą *npm install* aplanke *LisPlaylistV2/frontend.*
4. Aplanke *LisPlaylistV2/frontend* sukurkite arba papildykite failą *.env*.

   Failo turinys:

   REACT_APP_BROWSER=none

   REACT_APP_GOOGLE_AUTH_CLIENT_ID={Jūsų Google projekto client_id}

   REACT_APP_SPOTIFY_CLIENT_ID={ Jūsų Spotify projekto client_id}
5. Per terminalą paleiskite komandą *npm start* aplanke *LisPlaylistV2/frontend.*
6. Paleiskite *API* projektą per *Visual Studio*.

   Pastaba: projektas sukonfigūruotas, jog *React* aplikacija veikia ant *porto 3000, API* ant *porto 44339*. Norint pakeisti *portus* redaguokite failus */ListplaylistV2/ launchSettings.json*, */ListplaylistV2/Startup.cs*, */ListplaylistV2/frontend/src/config/GeneralConfig/GeneralConfig.js*.