SUSTAV ZA KONTROLE VERZIJA U KOLABORACIJSKOM OKRUŽENJU

Huzejrović, Albert

VISOKO UČILIŠTE ALGEBRA

ZAVRŠNI RAD

Sustav za kontrole verzija u kolaboracijskom okruženju

Albert Huzejrović

Zagreb, veljača 2020.

„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor, te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada".

U Zagrebu, 18.02.2020.

Albert Huzejrović

Predgovor

Ovim putem bih se htio zahvaliti svojem mentoru Aleksanderu Radovanu, koji je uložio svoje slobodno vrijeme u pomaganje meni pri planiranju i kreiranju ovog završnog rada.

Prilikom uvezivanja rada, Umjesto ove stranice ne zaboravite umetnuti original potvrde o prihvaćanju teme završnog rada kojeg ste preuzeli u studentskoj referadi

Sažetak

U ovome radu prezentira se programsko rješenje sustava za verzioniranje u kolaboracijskom okruženju. Verzioniranje je važan dio svakog programa ili proizvoda te svaki program ili proizvod prolazi kroz više iteracija u svojem životnom vijeku. Postoje dva tipa sustava za kontrole verzija: distribuirani i centralizirani. Git, Mercurial i Darcs su distribuirani sustavi za kontrole verzija te ih razlikuje način pohrane i težina korištenja. Desktop aplikacija je izvedena u WPF-u i pruža funkcionalnosti upravljanja repozitorijima i sinkronizaciju repozitorija s Google Cloud skladištem datoteka. Sustav sadrži dvije baze: bazu za podatke o korisnicima i repozitorijima i bazu za pohranu datoteka repozitorija. Baza za podatke izvedena je u Microsoft SQL Serveru, dok je pohrana datoteka izvedena u Google Cloudu. Web dio aplikacije izveden je u ASP.NET MVC i predstavlja mogućnost detaljnog pregleda povijesti projekta te mogućnost dodavanja drugih osoba u repozitorij. U primjeru korištenja predstavlja se tijek jednog projekta kroz verzije.

Ključne riječi: verzioniranje, C#, Cloud, softver, web, desktop

This bachelor's thesis will present a programming solution for software versioning in a collaborative environment.

Software versioning is an important part of every program & product, each of which undergo several iterations in their life cycle. There are two types of version control systems: distributed and centralized. Git, Mercurial and Darcs are distributed version control systems that are distinguished by storage methods and ease of access. The desktop app is run in WPF and provides repository management & sync functionalities with the Google Cloud file storage. The system contains two databases: User & repository information DB and the repository files DB. The information DB is run on the Microsoft SQL Server, while the data storage is run on Google Cloud. Web part of the app is run in ASP.NET MVC and features a detailed historical overview of the project, with the ability to add new users to the repository. This example presents a course of the project through versioning.

Key words: versioning, C#, Cloud, software, web, desktop

Sadržaj

1. Uvod

U svijetu audio produkcije proces izrade sadržaja često se odvija kroz proces kolaboracije. Audio producenti zajedničkim idejama i povratnim informacijama razmjenjuju svoju viziju završnog proizvoda te se time dobiva jedinstven sadržaj. S dolaskom digitalnog doba i razvoja tehnologije proizvodnja glazbe sve se više odvija u *out-of-the-box* pristupu. Sukladno s time počeli su se razvijati i brojni softveri za izradu audio sadržaja. Takav tip softvera naziva se digitalna audio radna jedinica (engl. *digital audio workstation*). To su alati koji omogućuju stvaranje raznih tipova sadržaja, a koriste se u branšama kao što su muzička industrija, radio industrija, filmska industrija, izrada audio efekt itd. Dizajnirani su bili s ciljem dobivanja sličnog okruženja za izradu audio sadržaja kao u audio studiju.

Kroz moderno i brzo korisničko sučelje nastoje što više olakšati pristup te pružaju sve funkcionalnosti potrebne da se ideja pretvori u gotov proizvod koji je spreman za izdavanje (engl. *release*).

Kada je u pitanju kolaboracijska funkcionalnost u softverima, nema postojećih integriranih rješenja. Kolaboracija se može postići direktnim slanjem projektnih datoteka između osoba ili slanjem izvedene (engl. *export*) datoteke audio sadržaja koje će onda primatelj moći uvesti unutar svojeg softvera i raditi modifikacije. Takav pristup je čest kada osobe koje kolaboriraju ne izrađuju sadržaj u istoj digitalnoj audio radnoj jedinci.

Zbog toga, kroz vrijeme počeli su se razvijati alati i softveri specijalizirani za područje razvoja audio sadržaja. Platforma Splice najrazvijenija je u tome smislu i jedna od brojnih servisa koji nude je i kolaboracijski servis.

U ovome radu predstavljeno je programsko rješenje izrade kolaboracijskog sadržaja. Inicijalne ključne funkcionalnosti sustava koje su postignute su prikaz promjena unutar projekta između verzija te lako i učinkovito korisničko sučelje. Sustav će biti prilagođen za rad s projektnim datotekama sustava za izradu audio sadržaja naziva FL Studio. Sustav će biti izveden u dvije okoline: web i desktop. Desktop aplikacije će obavljati zadaću kreiranja i modificiranja repozitorija, te vršenje njegove sinkronizacije s spremištem datoteka. Web aplikacija će pružati mogućnosti pregleda strukture projekta, njegove povijesti i omogućiti korisnicima dodavanje drugih korisnika u svoj repozitorij.

2. Sustavi za verzioniranje

Verzioniranje softvera (engl. *software versioning*) je proces pridavanja jedinstvenog imena ili jedinstvenog broja nekoj inačici softvera[1]. Tijekom životnog vijeka (engl. *life-cycle*) programa ili proizvoda dolazi do njegovih promjena, a verzioniranje omogućuje planirani i organizirani pristup praćenja tih promjena. Svaki program ili proizvod prolazi kroz više iteracija. Kroz iteracije u početku životnog vijeka inačice su više namijenjene dodavanju novih funkcionalnosti, dok su nakon više orijentirane u doradi istih.

Sustavi za verzioniranje objedinjuju proces verzioniranja u programsko rješenje koje nastoji pružati korisnicima mogućnost verzioniranja njihovih proizvoda ili programa.

2.1. Razvoj sustava za kontrole verzija kroz povijest

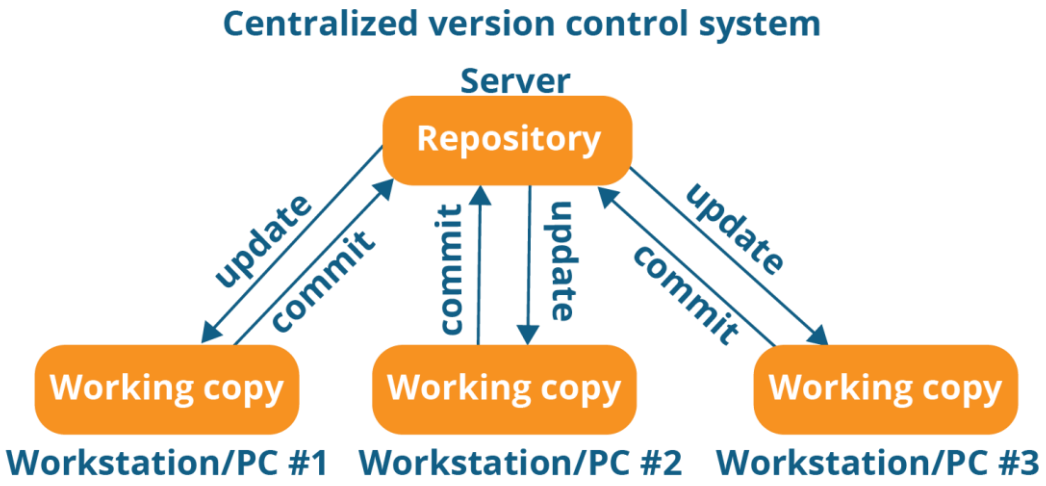
Prvi sustav kontrole verzija (engl. *version control system*) Source Code Control System bio je izrađen 1972. godine od strane Bell Labs. Pružao je osnovne funkcionalnosti praćenja promjena koda i drugih tekstualnih datoteka kroz životni tijek razvoja softvera. Nakon njega razvijen je Revision Control System 1982. godine. Zajedničko za oba sustava je da su bila temeljena na UNIX operacijskom sustavu i bili su namijenjeni za jednog korisnika. [2]

Postoje dvije kategorije sustava:

- Centralizirani sustav za kontrole verzija.
- Distribuirani sustav za kontrole verzija.

Centralizirani sustav za praćenje kontrole se bazira na ideji da postoji jedna centralna kopija, najčešće na serveru, i svi rade direktno izmjene na njoj.[3]

Slika 2.1. prikazuje tijek rada centraliziranog sustava za praćenje kontrola. Server sadrži kopiju projekta, te svi korisnici pristupaju kopiji koja se nalazi na serveru pri radu.



Slika 2.1 Način rada centraliziranog sustava za kontrole verzija[1]

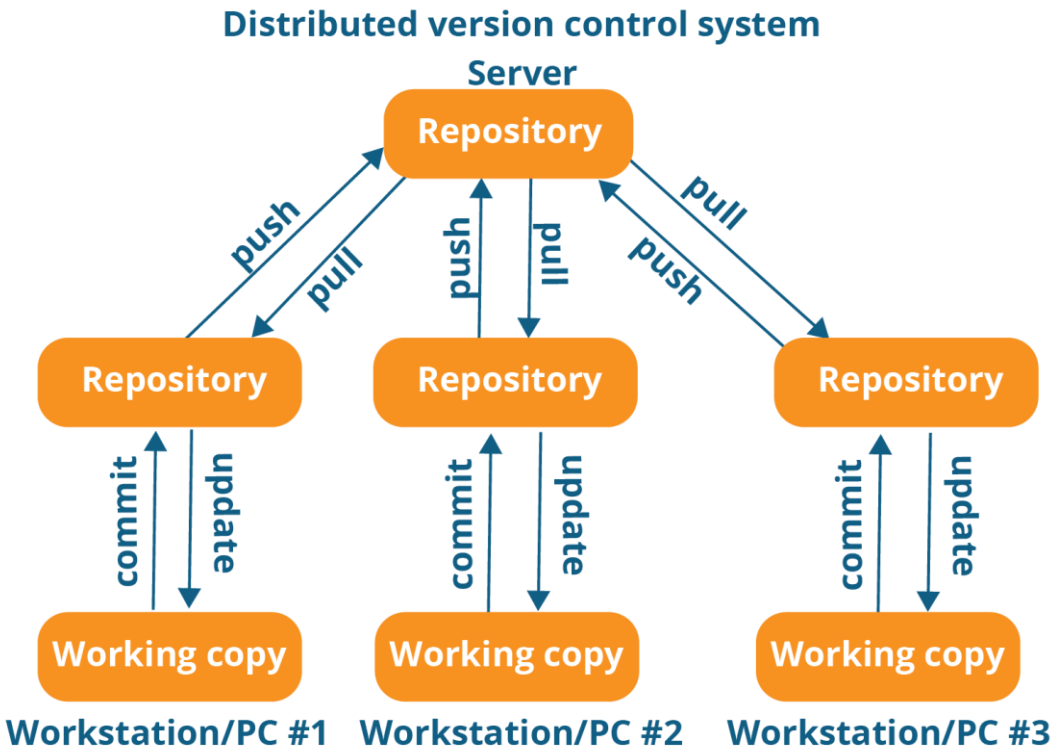
Najčešće će dijelovi koda koji se izmjenjuju biti zaključani za pristup, dok izmjena nije dovršena. Time se sprječava problem sinkronizacije različitih verzija u repozitorij.

Najpoznatiji centralizirani sustavi za kontrole verzija su CVS i Subversion.

Distribuirani sustavi za praćenje kontrole dijeli svoje kopije na centralni repozitorij i skup lokalnih skladišta. Na slici 2.2 moguće je vidjeti tijek rada s distribuiranim sustavom za praćenje kontrole. Programer s centralnog repozitorija povlači (engl. *pull*) projekt u svoj lokalni repozitorij. Lokalni repozitorij sadrži potpunu povijest sadržaja projekta. Programer sada može lokalno raditi na repozitoriju, vršiti spremanja (engl. *commit*) bez internetske veze. Kada se želi lokalni repozitorij spremiti u centralni, to se može učiniti s naredbom guranja (engl. *push*).

Centralnim repozitorijem najčešće upravljaju nositelji projekata (engl. *project maintainers*). Prije početka rada na repozitoriju, programeri kloniraju centralni repozitorij da bi stvorili lokalnu identičnu kopiju. Promjene na centralnom repozitoriju se periodički sinkroniziraju s lokalnim repozitorijem. Programeri kreiraju novi ogranak (engl. *branch*) i može početi raditi na tom ogranku. Nakon završetka, ogranak se sinkronizira u centralni repozitorij.

[1] https://medium.com/faun/centralized-vs-distributed-version-control-systems-a135091299f0



Slika 2.2. Način rada distribuiranog sustava za kontrole verzija[2]

2.2. Git, Mercurial i Darcs

Git, Mercurial i Darcs spadaju u kategoriju distribuiranog sustava za kontrole verzija. Sva tri sustava su besplatna i imaju sjedinjeni (engl. *merge*) model istodobnosti.

Git je daleko rasprostranjeniji u odnosu na ostale sustave za kontrole verzija. Koristi ga GitHub, najveća platforma za razvoj softvera. Glavne karakteristike koje njega razdvajaju od ostalih sustava su:

- Visoka brzina operacija.
- Moguće lokalno pristupiti svim starijim verzijama skladišta. Omogućuje mogućnost spremanja i pregleda sadržaja prethodnih verzija bez internetske veze.
- Laka i brza izrada ogranka.
- Visok integritet podataka.

[2] https://medium.com/faun/centralized-vs-distributed-version-control-systems-a135091299f0

- Promjena povijesti skladišta (redoslijed spremanja, sadržaje poruka, razdvajanje spremanja u više iteracija).

Prije sinkronizacije projekta u ogranak, datoteke u *Git*-u koje se žele ažurirati moraju se dodati kroz **pripremno područje** (engl. *staging area*). Nakon što su promjene dodane u pripremno područje, tek onda mogu se spremiti promjene. To omogućuje bolju kontrolu i planiranje sinkronizacija promjena, ali može biti zbunjujuće novim korisnicima.

Mercurial bi se mogao karakterizirati kao intuitivniji sustav od Git-a. Implementacija nekih od funkcionalnosti se razlikuju od Git-a. Primjerice, nemogućnost promjene povijesti skladišta. Nakon što je ogranak spojen (engl. *merge*) to označuje njegovo završno stanje. Takav pristup osigurava neporecivost i sigurnost podataka, ali može stvoriti probleme u slučaju spremanja pogreške.

Mercurial nudi mogućnost uvođenja dodataka (engl. *extension*) koji mogu pružati dodatne funkcionalnosti već postojećim komandama.

Darcs pruža slične funkcionalnosti vezane uz distribuirane sustave za kontrole verzija. U njemu postoje dva stanja: Radni direktorij (engl. *working directory*) i zakrpa (engl. *patch*). Podaci se iz radnog direktorija spremaju u zakrpu. Zakrpa sadržava spremljene podatke, a skladište se sastoji od skupa zakrpi.

Tablica 2.1. Usporedba sustava

	Git	**Mercurial**	**Darcs**
Godina nastajanja	2005.	2005.	2003.
Programski jezik	C	Python, C	Haskell
Razina kompleksnosti	Visoka	Srednja	Srednja
Brzina	Visoka	Srednja	Visoka
Način pohrane	Snapshot	Changeset	Changeset

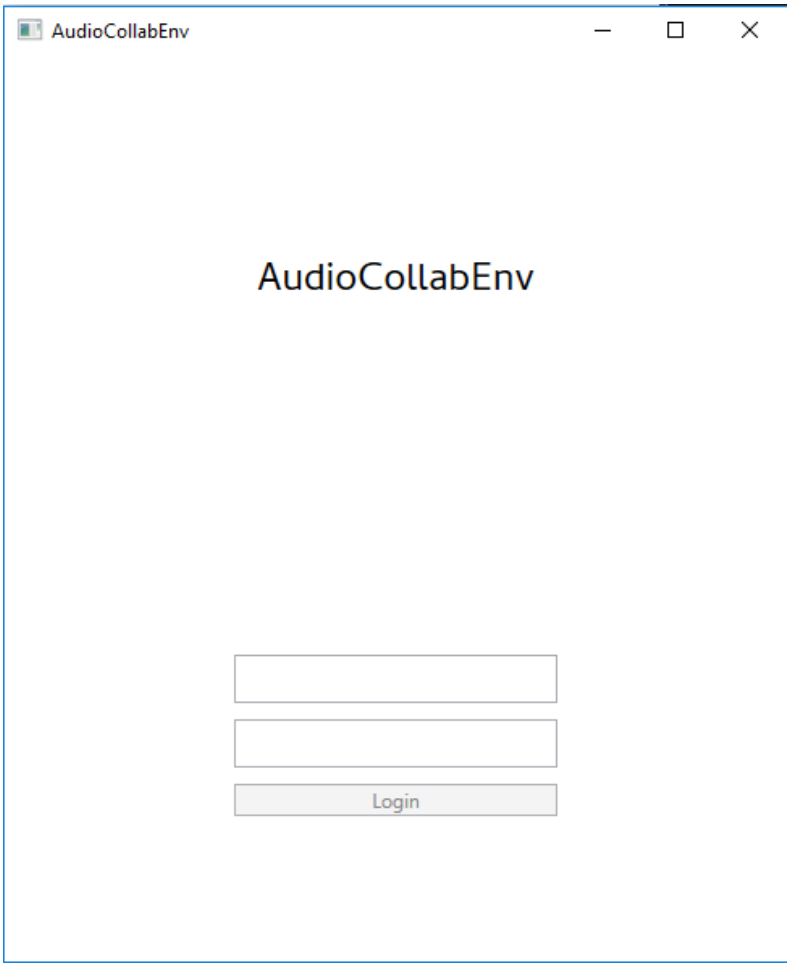
3. Arhitektura sustava

3.1. Desktop aplikacija

Desktop dio sustava izveden je u .NET radnom okruženju unutar *Windows Presentation Fundation*-a (*WPF*). Njegova primarna zadaća omogućiti korisnicima izradu repozitorija novog, vršenje njegove sinkronizacije, pregleda okvirnog pregleda sadržaja povijesti audio projekta u repozitoriju.

3.1.1. Opis funkcionalnosti

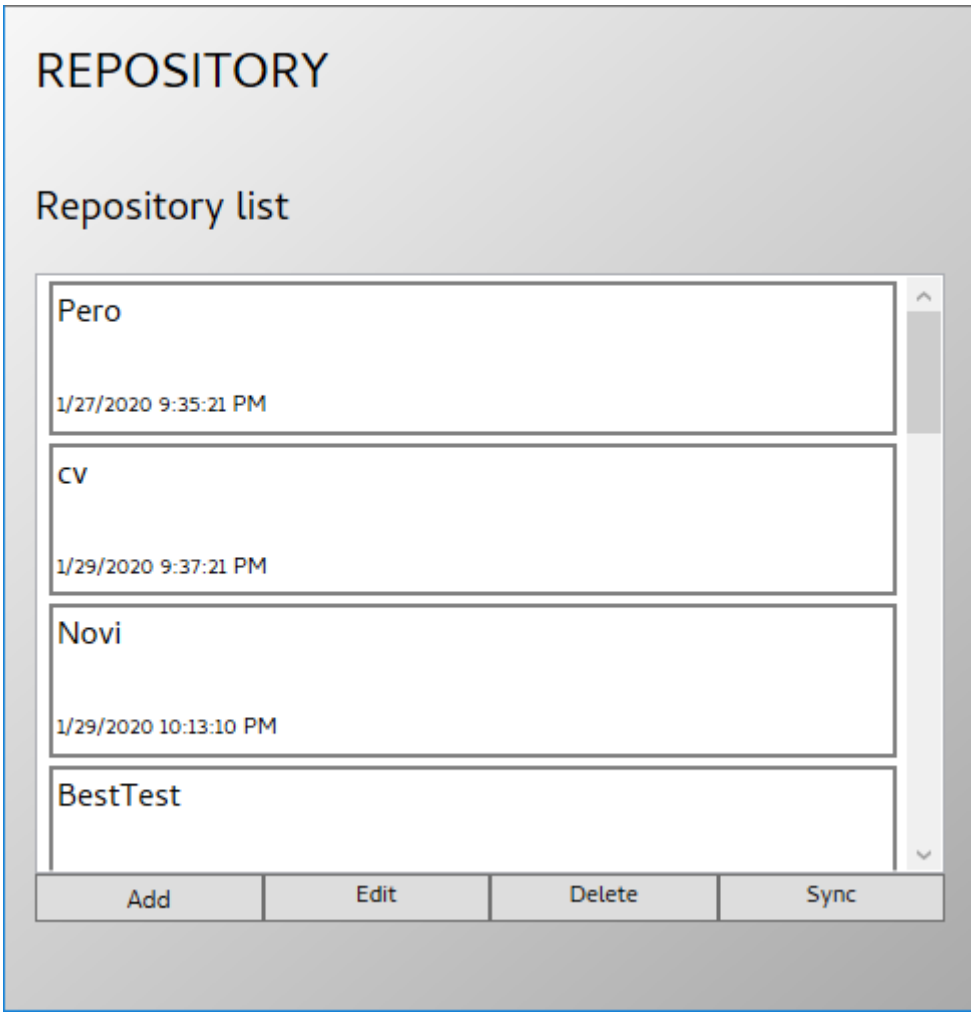
Pri ulasku u aplikaciju, korisnik će biti prezentiran s pogledom (engl. *view*) koji sadržava login formu. Na njoj se nalazi dva tekstna okvira za unos korisničkog imena i lozinke. Grafički prikaz pogleda prikazan je na slici broj 3.1.



Slika 3.1. Pogled za prijavu u desktop aplikaciju

Nakon popunjavanja podataka u tekstne okvire, gumb za prijavu postaje aktivan.

Ukoliko su uneseni podatci ispravni, korisnik je uspješno izvršio prijavu i prikazuje mu se glavni pogled desktop aplikacije koji je prikazan na slici 3.2.



Slika 3.2. Pogled za prikaz repozitorija

Glavni pogled sastoji se od popisa repozitorija od prijavljenog korisnika i skupa gumbova koji omogućavaju vršenje akcija nad označenim repozitorijem.

Iz desktop aplikacije dostupno je:

- Kreiranje novog repozitorija.
- Uređivanje naziva repozitorija.
- Brisanje repozitorija.
- Sinkronizacija sadržaja repozitorija sa serverom.

Program vrši spremanja i učitavanja repozitorija iz direktorija „*D:*\“. Prilikom sinkronizacije projekta, program će kreirati novu mapu naziva repozitorija unutar foldera te tamo postaviti projektnu datoteku. Nakon što je završena sinkronizacija repozitorija, korisnik može početi raditi na projektu. Kada korisnik završi sa svojim dijelom projekta, repozitorij je potrebno sinkronizirati na server tako da ostali članovi repozitorija mogu pristupiti novoj verziji.

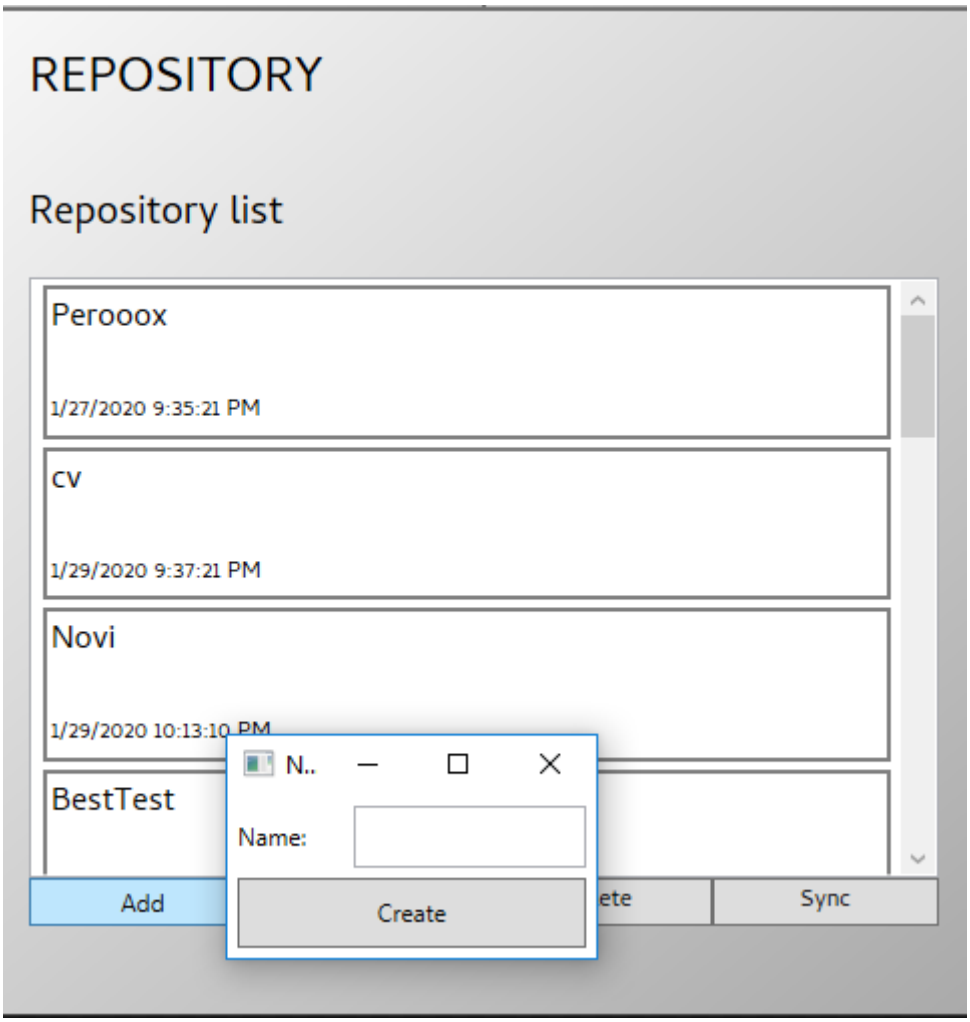
Ovisno o stanju u kojem se nalazi repozitorij, repozitoriju će biti dodijeljen pripadajući indikator koji obilježava njegovo stanje. Na slici 3.3. prikazan je indikator ažuriranog repozitorija. Indikatori su:

- Siva boja – repozitorij je kreiran, ali projekt još nije sinkroniziran s repozitorijem.
- Zelena boja – repozitorij je ažuriran na posljednju verziju projekta.
- Žuta boja – projekt je ažuriran od strane drugog korisnika unutar repozitorija, potrebno je izvršiti sinkronizaciju da se dohvati posljednja verzija projekta.

Slika 3.3. Indikator posljednje verzije repozitorija

Kreiranje novog repozitorija može se učiniti pritiskom na gumb *Add*. Pojavit će se novi prozor u koji je potrebno upisati željeno ime repozitorija i stisnuti na gumb *Create*.

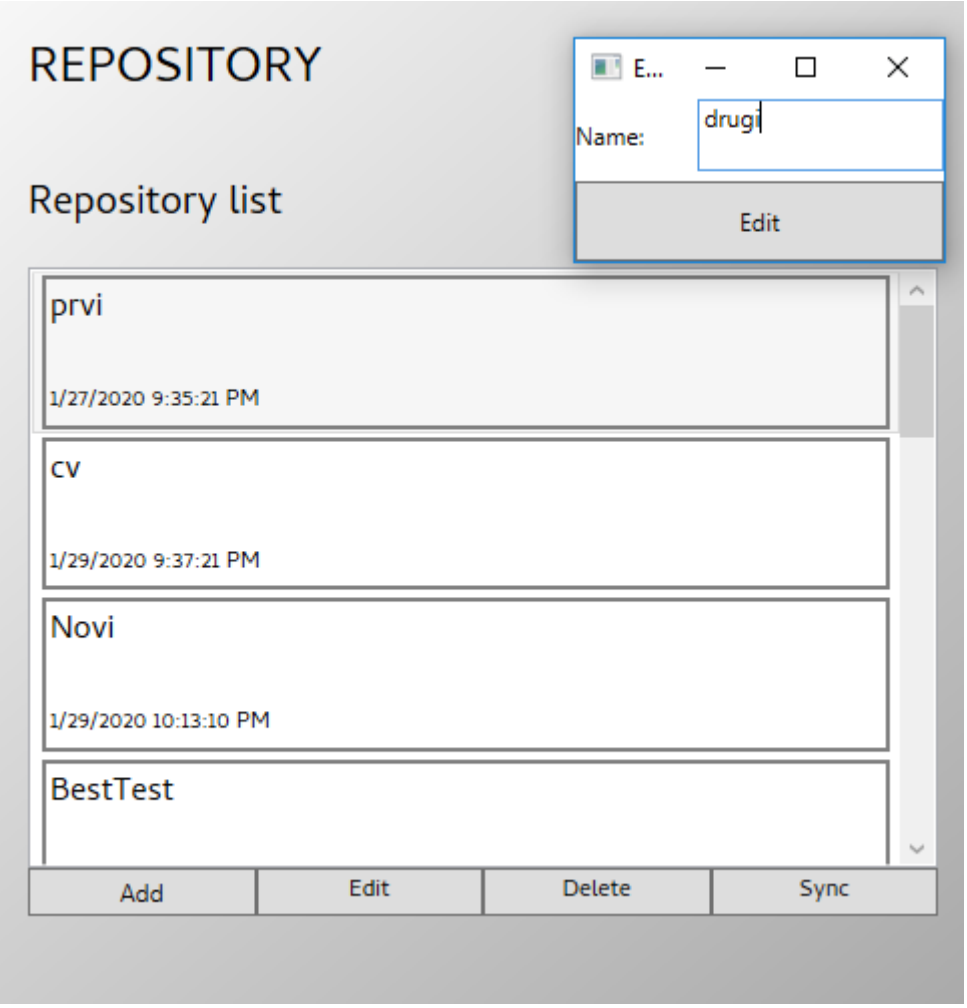
Na slici 3.4. vidljiv je prozor za kreiranje repozitorija. Nakon toga sustav će izvršiti kreiranje novog repozitorija i postaviti ga u listu trenutnih repozitorija.



Slika 3.4. Dodavanje novog repozitorija.

Uređivanje repozitorija dostupno je na gumb *Edit* i slično kao i s akcijom za kreiranje repozitorija, otvori se novi prozor u koji se upisuje novi naziv repozitorija.

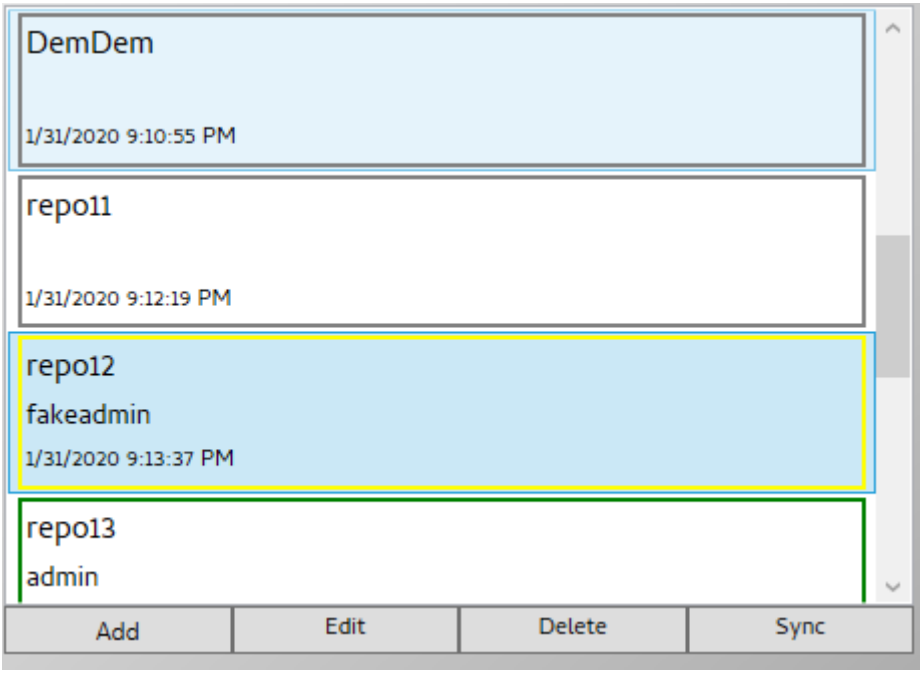
Na slici 3.5. vidljiv je proces promjene imena repozitorija.



Slika 3.5. Promjena naziva repozitorija

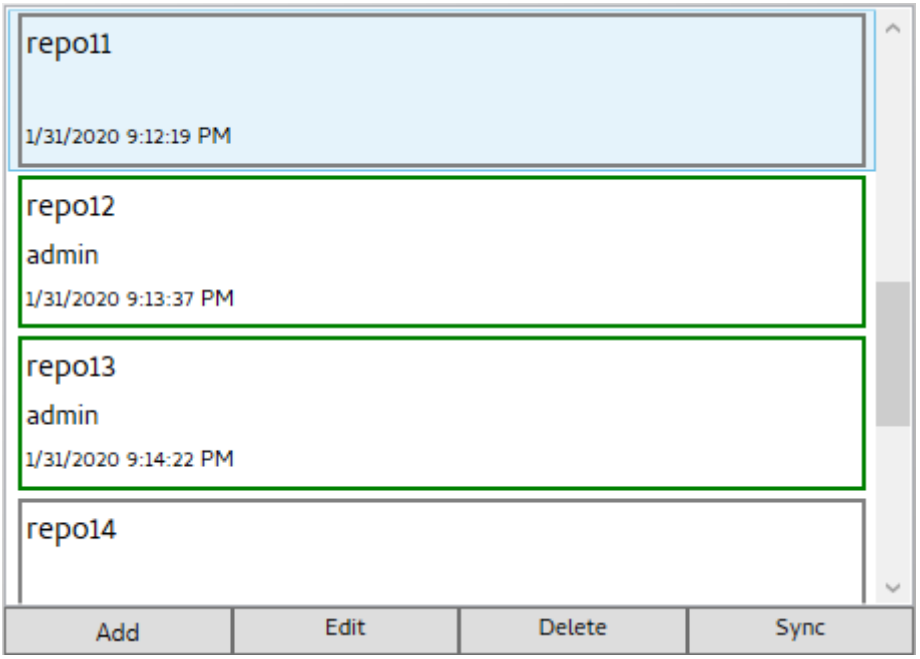
Kada se pritisne na gumb *Sync*, izvrši se provjera verzije projekta. Ukoliko je projekt najnovija verzija, ništa se ne događa. U suprotnome, projekt se postavlja u *Google Cloud* spremište podatka i ažurira se verzija repozitorija.

Na slici 3.6 vidljiva je situacija u kojem je drugi korisnik ažurirao stanje projekta te se prijavljenom korisniku prikazao žuti indikator.



Slika 3.6. Potrebno ažuriranje repozitorija

Korisnik sada može putem desktop ili web aplikacije pogledati sadržaj nove verzije, te odlučiti želi li on izvršiti sinkronizaciju na novu verziju projekta. Na slici 3.7 vidljiv je rezultat akcije pritiska na gumb sinkronizacije.

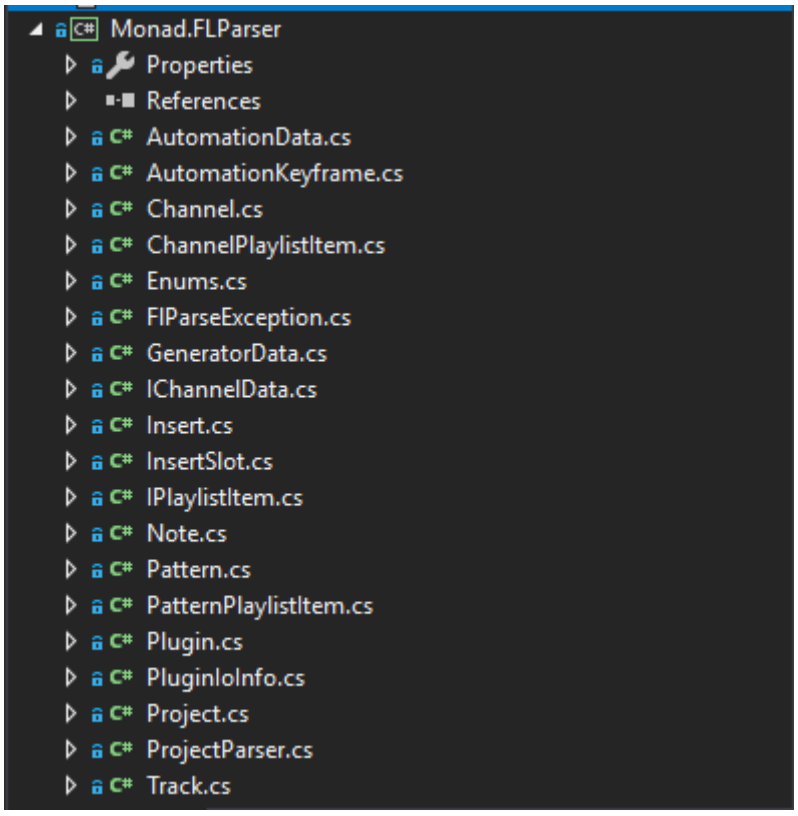


Slika 3.7. Rezultat akcije sinkroniziranja

3.1.2. Učitavanje i procesiranje audio datoteka

Veliki dio sustava temelji se na obradi i prikazu podataka iz digitalne radne jedinice. Ovisno o softveru, postoje različiti tipovi datotečnih nastavaka. Sustavi spremaju sve informacije o stanju projekta u datoteku, tako da se pri učitavanju može nastaviti s daljnjim radom.

Budući da je sustav specijaliziran za rad s *FL* Studiom, potrebno je učitati i procesirati podatke datotečnog nastavka *.flp* (engl. *FL project*). Podaci projekta se spremaju u binarnom obliku, te ih je potrebno iščitati i ubaciti u sustav. Za tu potrebu koristit će se vanjska biblioteka *Monad FLParser*. Ona zadovoljava osnovne funkcionalnosti čitanja datoteka. Za njezino korištenje prvo ju je potrebno preuzeti i dodati kao referencu unutar projekta. Sama biblioteka osim funkcionalnosti čitanja već dolazi s klasama koje označavaju elemente unutar projekta. Na slici 3.8. prikazana je struktura biblioteke.



Slika 3.8. Struktura biblioteke

Za početak korištenja potrebno je pozvati statičku metodu `Load` klase `Project`. Metoda kao parametar prima način učitavanja, i zastavicu `verbose` koja s kojom se može podesiti razina učitavanja – normalna ili opširna.

Postoje tri načina učitavanja projektne datoteke:

- Iz putanje (engl. *path*).
- Iz toka (engl. *stream*).
- Iz objekta binarnog čitača (engl. *binary reader*).

Prilikom učitavanja biblioteka čita (engl. *parse*) binarnu datoteku i vraća objekt koji je tipa `Project`. Kod 3.2 prikazuje isječak koda za čitanje binarne datoteke.

```
private void ParseHeader(BinaryReader reader)
{
    if (Encoding.ASCII.GetString(reader.ReadBytes(4)) != "FLhd")
      throw new FlParseException("Invalid magic number",
      reader.BaseStream.Position)
// header + type
            var headerLength = reader.ReadInt32();
            if (headerLength != 6)
                  throw new FlParseException($"Expected header length 6,
                  not {headerLength}", reader.BaseStream.Position);

            var type = reader.ReadInt16();
            if (type != 0) throw new FlParseException($"Type {type} is
            not supported", reader.BaseStream.Position);

            // channels
            var channelCount = reader.ReadInt16();
            if (channelCount < 1 || channelCount > 1000)
                  throw new FlParseException($"Invalid number of
                  channels: {channelCount}", reader.BaseStream.Position);
            for (var i = 0; i < channelCount; i++)
            {
                  _project.Channels.Add(new Channel { Id = i, Data = new
                  GeneratorData() });
            }
             // ppq
             _project.Ppq = reader.ReadInt16();
             if (_project.Ppq < 0) throw new Exception($"Invalid PPQ:
            {_project.Ppq}");
        }
```

Kod 3.1. Isječak iz čitača binarne datoteke

Klasa `Project` sadrži informacije o projektu koji je učitan. Prikazana je na kodu broj 3.2.

```
public int MainVolume { get; set; } = 300;
public int MainPitch { get; set; } = 0;
public int Ppq { get; set; } = 0;
public double Tempo { get; set; } = 140;
public string ProjectTitle { get; set; } = "";
public string VersionString { get; set; } = "";
public int Version { get; set; } = 0x100;
public List<Channel> Channels { get; set; } = new List<Channel>();
public Track[] Tracks { get; set; } = new Track[MaxTrackCount];
public List<Pattern> Patterns = new List<Pattern>();
public Insert[] Inserts { get; set; } = new Insert[MaxInsertCount];
public bool PlayTruncatedNotes { get; set; } = false;
public string FileVersion { get; set; }
```

Kod 3.2. Klasa `Project`

Osim osnovnih informacija o projektu kao što su tempo, naziv i volumen, klasa sadrži skup traka, kanala, uzoraka i umetaka (engl. *inserts*).

3.1.3. Implementacija funkcionalnosti

Windows Presentation Foundation (*WPF*) prozor se sastoji od dvije komponente: korisničko sučelje definira datoteka nastavka *.xaml*, dok pripadajuća datoteka istog naziva prozora *.cs* definira programsku logiku prozora.

Unutar *.xaml* datoteke nalazi se skup čvorova (engl. *node*). Naziv čvora opisuje tip kontrole koja će se prikazati na korisničkom sučelju. Osim naziva, kontrolu opisuju i ostala svojstva (engl. *properties*) s kojima je moguće promijeniti izgled kontrole i ponašanje same kontrole, ukoliko se želi.

Kod broj 3.3. *.xaml* dateoteku koja sadržava prikaz liste repozitorija iz baze podataka. Osim glavnog čvora vidljive su i neke od mogućnosti uređivanja načina ponašanja kontrola.

```
<ListBox
      Grid.Column="0"
      x:Name="ProjectList"
      HorizontalContentAlignment="Stretch"
```

```
      ScrollViewer.VerticalScrollBarVisibility="Visible"
      MaxHeight="300">
<ListBox.ContextMenu>
      <ContextMenu>
            <MenuItem Header="View details" x:Name="ContextMenuDetails"
            Click="ContextMenuDetails_Click"/>
      </ContextMenu>
      </ListBox.ContextMenu>
</ListBox>
```

Kod 3.3. .xaml prikaz liste repozitorija

Unutar *.cs* datoteke moguće je pristupiti instancama kontrola preko identifikatora te koristiti razne metode na njima. Osim toga moguća je i pretplata na događaje (engl. *event*) koji će se izvršiti nakon okidača na prozoru, primjerice klik miša.

```
<Page x:Class="AudioCollab_DesktopApp.Pages.MenuPage"
xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:local="clr-namespace:AudioCollab_DesktopApp.Pages"
mc:Ignorable="d"
d:DesignHeight="450" d:DesignWidth="800"
Title="MenuPage">
<Grid></Grid>
</Page>
```

Kod 3.4. Osnovna .xaml datoteka

```
namespace AudioCollab_DesktopApp
{
      public partial class MainWindow : Window
      {
            public MainWindow()
            {
                  InitializeComponent();
                  SetPage(PageList.LoginPage);
            }
            public void SetPage(Page page)
```

```
            {
                  this.Content = page;
            }
      }
}
```

Kod 3.5. *.cs* datoteka početnog prozora

Kada korisnik uđe u aplikaciju, unutar konstruktora početnog prozora poziva se metoda `SetPage` pomoću koje se kroz kod postavlja prozor koji će se prikazati korisnik. Inicijalno mu se postavlja login stranica.

Klasa `PageList` koja je vidljiva na prikazu koda broj 3.5. sadrži statička svojstva koja služe za promjenu sadržaja prozora. Na kodu broj 3.6. vidljivo je svojstvo za login stranicu.

```
private static LoginPage _loginPage;
public static LoginPage LoginPage
{
      get
      {
            if(_loginPage == null)
            {
                  _loginPage = new LoginPage();
            }
            return _loginPage;
      }
}
```

Kod 3.6. *.cs* svojstvo koje sadrži stranicu za login

Nakon što korisnik unese email i lozinku u pripadajuće tekstualne okvire koji su vidljivi na slici broj 3.1., poziva se metoda `LoginUser` pomoću koje se vrši autentikacija korisnika.

Unutar te metode kreira se instanca `UserAccountService` klase. To je servis kojemu je zadaća iz primljenih korisničkih podataka provjeriti njihovu točnost i vratiti programu objekt prijavljenog korisnika.

```
public class UserAccountService : IAccountService
{
      public IIdentity ValidateUser(string username, string password)
      {
            IIdentity identity = null;
```

```
            IAuthenticationService uaService =
            AuthenticationFactory.Build(new UserAuthenticationService());
            identity = uaService.Authenticate(username, password);
            return identity;
      }
}
```

Kod 3.7. Programski kod servisa za prijavu korisnika

Metoda `ValidateUser` prima korisničke podatke, kreira instancu autentikacijskog servisa te njemu prosljeđuje podatke. Autentikacijski servis je u sloju pristupa podataka te se njegov posao sastoji od traženja korisnika u bazi i provjere njegove lozinke. Ukoliko su podatci točni kreira se instanca `ClaimsIdentity` klase koja će predstavljati prijavljenog korisnika u aplikaciji.

Kod za autentikacijski servis prikazan je na rednom broju 3.8.

```
public class UserAuthenticationService : IAuthenticationService
{
      public IIdentity Authenticate(string username, string verifier)
      {
            ClaimsIdentity identity = null;
            if (username == null || verifier == null) return null;
            List<AudioCollab_DAL.Database.Account> accounts =
            DbContextFactory.Build().GetUsers().ToList();
            AudioCollab_DAL.Database.Account userAccount =
            accounts.FirstOrDefault(account => account.Username ==
            username);
            if (userAccount == null) return null;
            if (userAccount.Password == verifier)
            {
                  List<Claim> userClaims = new List<Claim>
                  {
                        new Claim("id",
                        userAccount.IDAccount.ToString()),
                        new Claim("username", username)
                  };
                  identity = new ClaimsIdentity(userClaims,
                  AuthenticationTypes.Password);
            }
            return identity;
```

```
        }
}
```

Kod 3.8. Autentikacijski servis

Ukoliko su podaci ispravni, kreira se lista tvrdnji (engl. *claim*) koji opisuju korisnika. Iz te liste kreira se instanca `ClaimsIdentity` klase koja se vraća nazad u servis, a zatim nazad do korisničkog sučelja.

Ovim pristupom razdvajanja procesa prijave korisnika u zasebne servise dobiva se na modularnosti same aplikacije. Ukoliko se, primjerice, u budućnosti želi drugačijim pristupom vršiti prijava korisnika, sve što je potrebno učiniti je dodati novu klasu koja implementira `IAccountService` sučelje (engl. *interface*), i tamo dodati drugačiji način validacije korisnika.

Isto tako, budući da `UserAccountService` koristi `IAuthenticationService`, lako je moguće promijeniti ili proširiti autentikacijski servis bez promjene `UserAccountService` klase.

```
private void LoginUser()
{
        UserAccountService service = new UserAccountService();
        UserSession.UserSession.CurrentUser = service.ValidateUser(Email,
        Password);
        PageList.MainWindow.SetPage(PageList.MainPage);
}
```

Kod 3.9. Prijava korisnika u sustav

Na prikazu koda broj 3.9. vidljiva je metoda koja prijavljuje korisnika u sustav. Nakon prijave pomoću klase `PageList` postavlja se stranica koja sadrži listu repozitorija. Korisnik se sprema u statičku metodu `CurrentUser` klase `UserSession`.

Pri inicijalizaciji stranice liste repozitorija dohvaća se popis repozitorija u kojima je uključen prijavljen korisnik. Sve akcije vezane uz dohvaćanje, dodavanje, uređivanje ili brisanje repozitorijskih podataka vrši se preko `RepositoryService` klase.

```
private List<RepositoryViewModel> GetRepositoryViewModels()
{
        AudioCollab_BL.Repository.RepositoryService repositoryService = new
        AudioCollab_BL.Repository.RepositoryService();
```

```
        ClaimsIdentity userIdentity = UserSession.UserSession.CurrentUser
        as ClaimsIdentity;
        string id = userIdentity.Claims.FirstOrDefault(c => c.Type ==
        "id").Value;
        IEnumerable<Repository> repositories =
        repositoryService.GetRepositoriesForAccount(int.Parse(id));
        List<RepositoryViewModel> repositoryViewModel = new
        List<RepositoryViewModel>();
        foreach (Repository repo in repositories)
        {
              bool isLatest = false;
              string lastAccountModified = "";
              AccountRepositoryVersion latestAccountRepositoryVersion =
              repositoryService.GetLastAccountRepositoryVersion(repo.IDRepo
              sitory);
              if(latestAccountRepositoryVersion != null)
              {
                    int? userAccountRepositoryVersion =
                    repositoryService.GetProjectVersionForAccount(repo.IDRe
                    pository, int.Parse(id));
                    lastAccountModified =
                    repositoryService.GetAccount(latestAccountRepositoryVer
                    sion.AccountID).Username;
                    if (userAccountRepositoryVersion ==
                    latestAccountRepositoryVersion.ProjectVersion)
                    {
                          isLatest = true;
                    }
              }
              repositoryViewModel.Add(new RepositoryViewModel { Repository
              = repo, IsLatestVersion = isLatest, LastAccountModified =
              lastAccountModified });
              }
              return repositoryViewModel;
}
```

Kod 3.10. Metoda dohvaćanja liste repozitorija

Nakon što se dohvati lista repozitorija za korisnički račun, kreira se lista `RepositoryViewModel` klase koja predstavlja model vizualnog prikaza repozitorija.

Primjerice, oznaka da li je repozitorij ažuriran na posljednju verziju ili ne. Za svaki repozitorij u listi repozitorija vrši se dodatna provjera da se utvrdi koji račun sadrži posljednju verziju projektne datoteke.

Ukoliko ulogirani korisnik odgovara zadnjem korisniku koji je ažurirao stanje projekta, postavlja se zastavica `isLatest`, te se prikladno tome ažurira vizualni prikaz objekta u listi.

Nakon što se ispuni lista `RepositoryViewModel` klase, postavlja se kao izvor podataka listi na korisničkom sučelju. Postavljanje izvora i cijelu metodu moguće je vidjeti na prikazu koda broj 3.11.

```
private void SyncRepo_Click(object sender, RoutedEventArgs e)
{
      Repository selectedRepository = (ProjectList.SelectedItem as
      RepositoryViewModel).Repository;
      if(selectedRepository != null)
      {
            UploadService uploadService = new UploadService();
            FileStorageService service = new FileStorageService();
            if (service.HasFileChanged(selectedRepository,
            UserSession.UserSession.RepoPath + "\\" +
            selectedRepository.Name + "\\"))
            {
                  ClaimsIdentity userIdentity =
                  UserSession.UserSession.CurrentUser as ClaimsIdentity;
                  string id = userIdentity.Claims.FirstOrDefault(c =>
                  c.Type == "id").Value;
                  bool result =
                  uploadService.SyncRepo(UserSession.UserSession.RepoPath
                  , int.Parse(id), selectedRepository);
             }
             SetItemSource();
      }
}
```

Kod 3.11. Kod sinkronizacije repozitorija

3.1.4. Sinkronizacija projekata

Da bi sustav uspješno i točno kreirao novu verziju projekta, potrebno je izvršiti njegovu sinkronizaciju.

Prilikom sinkronizacije, uzima se označeni repozitorij izliste repozitorija. Nakon toga se kreiraju instance klasa `UploadService i FileStorageService.`

- `UploadService` klasa je implementirana kao servis koji komunicira s Google Cloud i vrši sinkronizaciju projekta.
- `FileStorageService` sadrži metode za upravljanjem folderima na Google Cloudu i metodu za provjere jesu li se datoteke promijenile.

Poziva se metoda `HasFileChanged` na servisu koji uspoređuje ključeve stare i nove datoteke da se utvrdi jesu li iste ili ne.

Ukoliko se dogodila promjena između dvije verzije, potrebno je ažurirati datoteku.

Dohvaća se prijavljeni korisnik u aplikaciji i poziva se metoda `SyncRepo` klase `UploadService.` Metoda je prikazana na prikazu koda broj 3.12.

```
public bool SyncRepo(string repoPath, int accountId,
AudioCollab_DAL.Database.Repository repository)
{
      FileStorageContext context = new FileStorageContext();
      string filePath = repoPath + "\\" + repository.Name + "\\";
      repository.ProjectVersion = repository.ProjectVersion + 1 ?? 1;
      RepositoryService repositoryService = new RepositoryService();
      repositoryService.UpdateVersion(repository);
      repositoryService.InsertProjectVersionForAccount(repository.IDRepos
      itory, accountId, repository.ProjectVersion);
      return context.UploadRepository(repository, filePath);
}
```

Kod 3.12. Metoda SyncRepo klase UploadService

Metoda instancira klasu `FileStorageContext` kojoj je zadaća pristupiti Google Cloud bazi podataka i vršiti akcije na njoj. Prije ažuriranja repozitorija u Google Cloudu ažurira se stanje repozitorija u bazi podataka.

```
public bool UploadRepository(Repository repository, string filePath)
{
```

```
byte[] fileSum = FileSumGenerator.CalculateSum(filePath +
repository.Name + ".flp");
using (var context = new Database_v2())
{
      Repository dbRepository =
      context.Repositories.FirstOrDefault(repo => repo.IDRepository
      == repository.IDRepository);
      dbRepository.LastModifiedSum = fileSum;
      context.SaveChanges();
}
bool hasUplodaded = false;
var bucket = GetStorageClient();
var credential = GoogleCredential.FromFile(_jsonPath);
var storageClient = StorageClient.Create(credential);
string projectFilePath = repository.IDRepository + "/" +
repository.ProjectVersion + "/" + repository.Name + ".flp";
IDictionary<string, string> fileMetadata = new
Dictionary<string,string>
{
      { "version", repository.ProjectVersion.ToString() }
};
var newObject = new Google.Apis.Storage.v1.Data.Object()
{
      Bucket = bucket.Name,
      Name = projectFilePath,
      Metadata = fileMetadata
};
FileStream fileStream = null;
try
{
      fileStream = new FileStream(filePath + repository.Name +
      ".flp", FileMode.Open, FileAccess.Read, FileShare.Read);
      storageClient.UploadObject(newObject, fileStream);
      hasUplodaded = true;
      return hasUplodaded;
}
catch (Exception)
{
      return hasUplodaded;
}
```

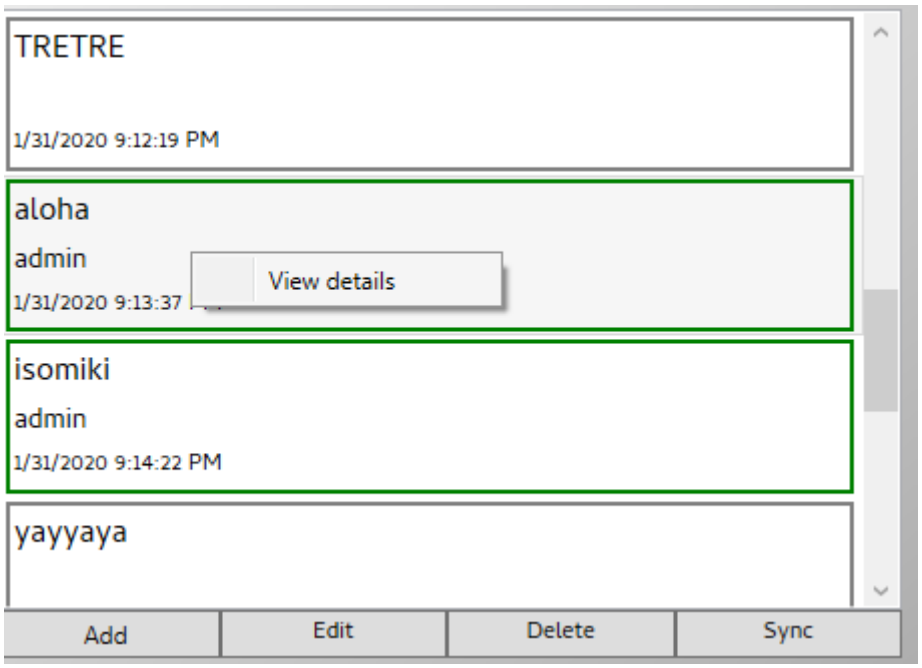
```
        finally
        {
                if (fileStream != null)
                {
                        fileStream.Dispose();
                }
        }
}
```

Kod 3.13. Metoda za prijenos datoteke u Google Cloud

U kodu broj 3.13. prikazana je metoda prijenosa datoteke u Google Cloud. Na početku metode, pomoću klase korisnosti (engl. *utility*) `FileSumGenerator` izračunava se suma za danu datoteku te je se poslije toga sprema u *SQL* bazu podataka. Ona je potrebna za usporedbu da li su datoteka u Google Cloudu i nova datoteka koja se želi sinkronizirati identične. Nakon toga izrađuje se novi Google Cloud objekt te se prenosi u skladište datoteka (engl. *file storage*).

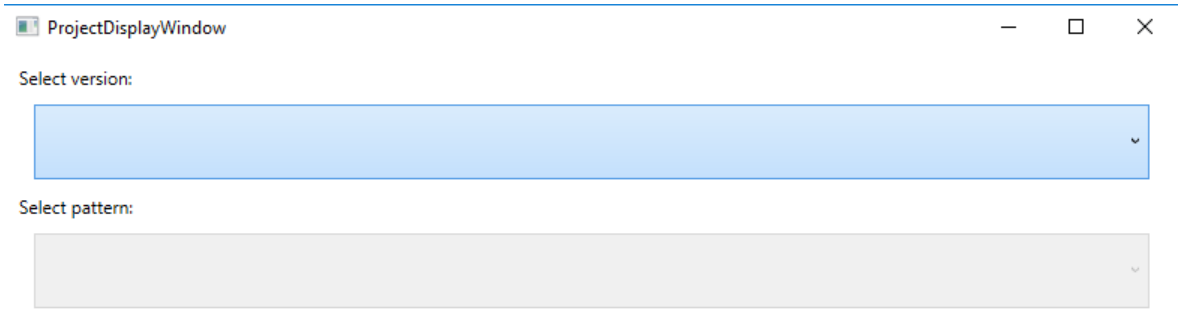
3.1.5. Vizualizacija podataka

Unutar desktop aplikacije vizualizacija podataka će biti izvedena kroz grafički prikaz u *.xaml* prozoru. Korisnik će otvaranjem *context menu*-a moći odabrati opciju *View details* pomoću koje se može pristupiti pregledu povijesti repozitorija.



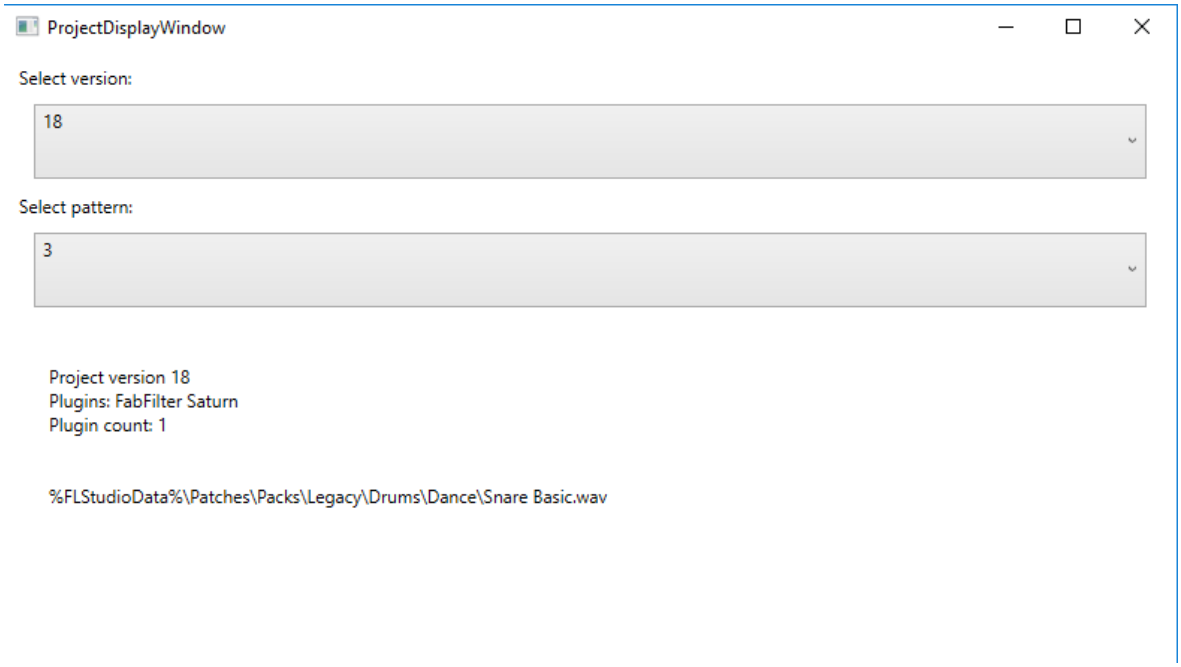
Slika 3.9. Kontekst meni

Kada se odabere opcija, otvara se novi prozor u kojem se može podesiti verzija projekta za koje se žele vidjeti detalji.



Slika 3.10. Forma za vizualni prikaz podataka

Nakon odabira željene verzije, dobije se informacija koji *plugin-ovi* su uključeni u ovu verziju projekta. Na slici broj 3.10. prikazan je primjer odabira opcija.



Slika 3.11. Primjer vizualizacije

Kada se učitaju informacije, učita se i dodatni kombinirani okvir (engl. *combo box*) u kojem se može odabrati uzorak (engl. *pattern*) iz projekta te se dodatno ispisuje njegov sadržaj ispod tekstualnog dijela verzije projekta.

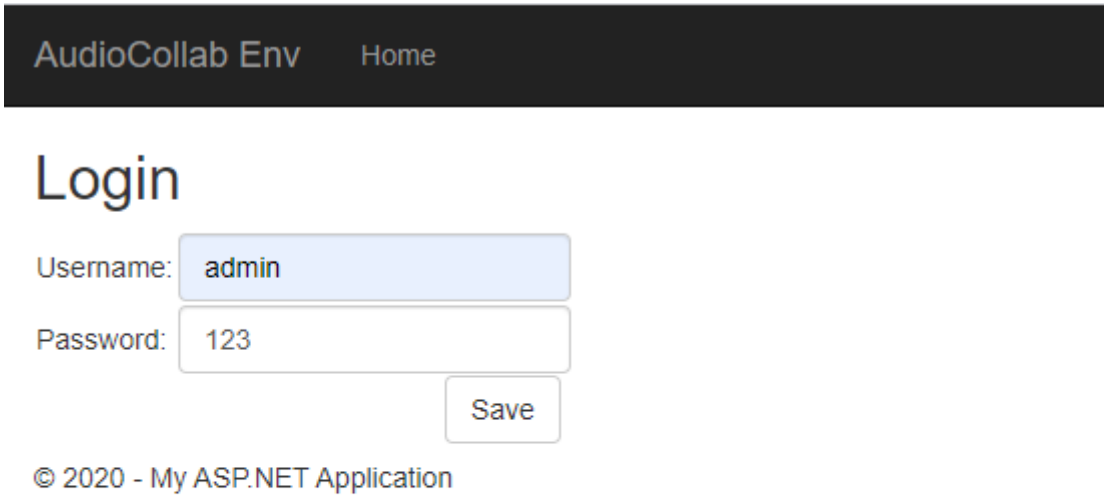
3.2. Web aplikacija

Web aplikacija izvedena je u *ASP.NET MVC* i pruža funkcionalnost za pregled repozitorija u kojem se korisnik nalazi, funkcionalnost za dodavanje drugih osoba u repozitorij, i mogućnost pregleda strukture projekta.

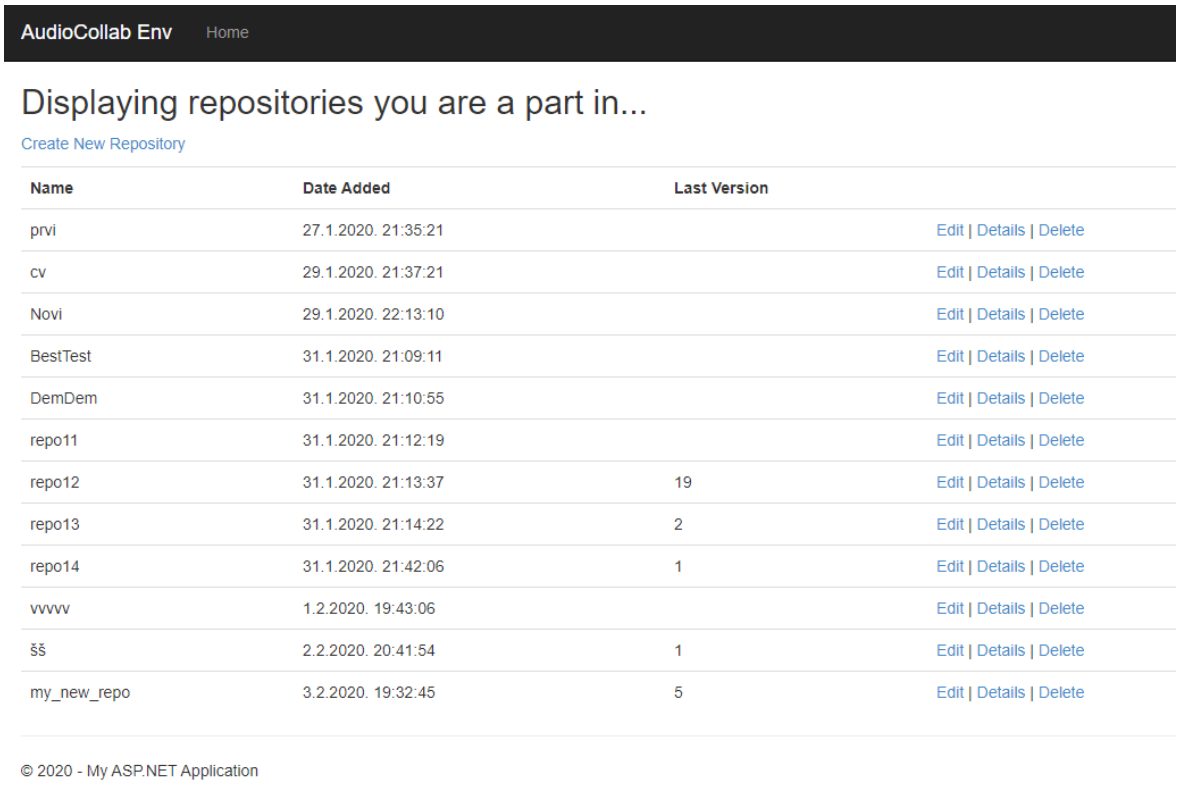
3.2.1. Opis funkcionalnosti

Pri ulasku na stranicu, korisnika će sustav preusmjeriti na stranicu za login gdje će morati unijeti svoje podatke da se prijavi u sustav. Stranica za prijavu prikazana je na slici broj 3.12. Nakon što se unesu podatci, vrši se provjera točnosti unesenih podataka.

Ako je korisnik pronađen i lozinka odgovara, korisnika se prosljeđuje na početnu stranu liste repozitorija. Prikaz izgleda liste repozitorija vidljiva je na slici broj 3.13.



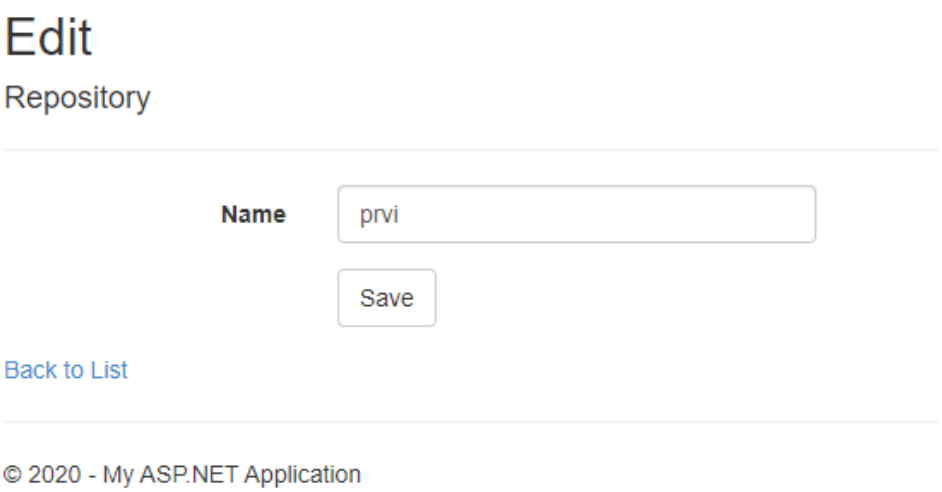
Slika 3.12. Prijava korisnika u sustav



Slika 3.13. Stranica popisa repozitorija

Pritiskom na akciju uređivanja (engl. *edit*) otvara se nova stranica u kojoj je moguće preimenovanje naziva repozitorija.

Na slici 3.14. prikazana je stranica za uređivanje repozitorija.



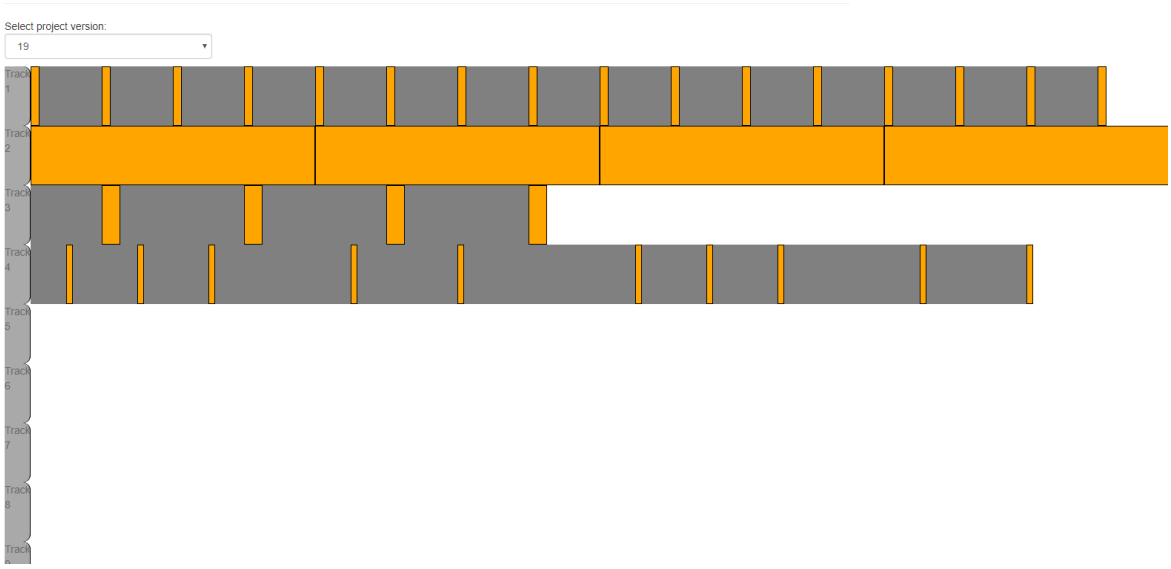
Slika 3.14. Stranica za uređivanje naziva repozitorija

Pritiskom na akciju detalja (engl. *details*) korisnika se preusmjerava na stranicu pregleda sadržaja repozitorija kroz povijest.

Na stranici korisnik je prezentiran s padajućim izbornikom u kojem se nalaze sve verzije projekta koje su sinkronizirane u Google Cloud skladištu datoteka.

Nakon što korisnik odabere verziju koju želi prikazati, stranica će vratiti vizualnu strukturu sadržaja koji čine taj projekt.

Na slici 3.15. prikazana je stranica detalja.

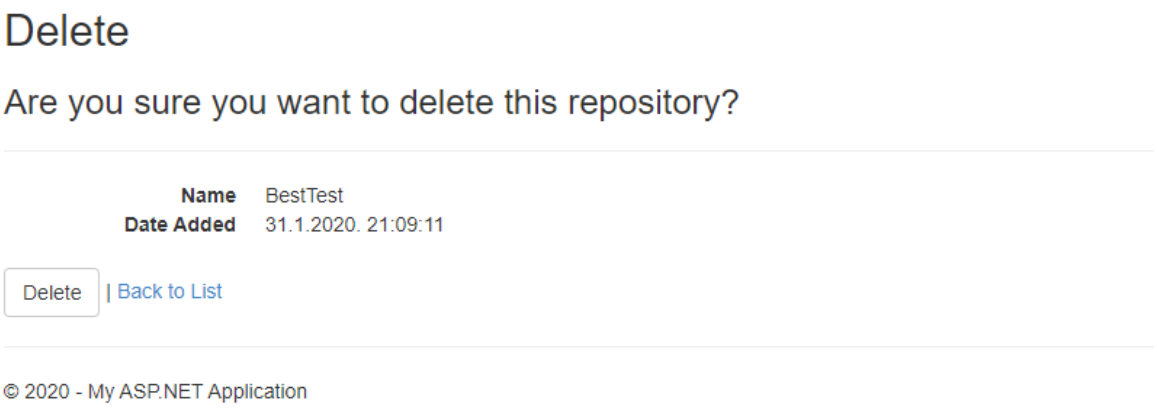


Slika 3.15. Grafički prikaz sadržaja projekta

Struktura toga istog projekta unutar programa FL Studio se može vidjeti na slici broj 3.16.

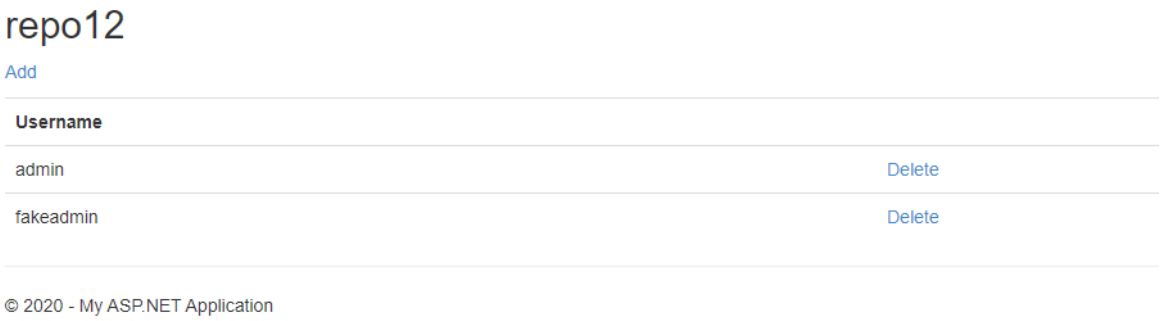
Slika 3.16. Struktura projekta

Pritiskom na akciju brisanja (engl. *delete*) korisnika se usmjeri na stranicu brisanja, koja se sastoji od pregleda svojstva odabranog repozitorija i gumba za brisanje. Nakon pritiska gumba, sav sadržaj repozitorija se briše. Slika 3.16. prikazuje sadržaj stranice za brisanje repozitorija.



Slika 3.17. Prikaz sadržaja stranice za brisanje

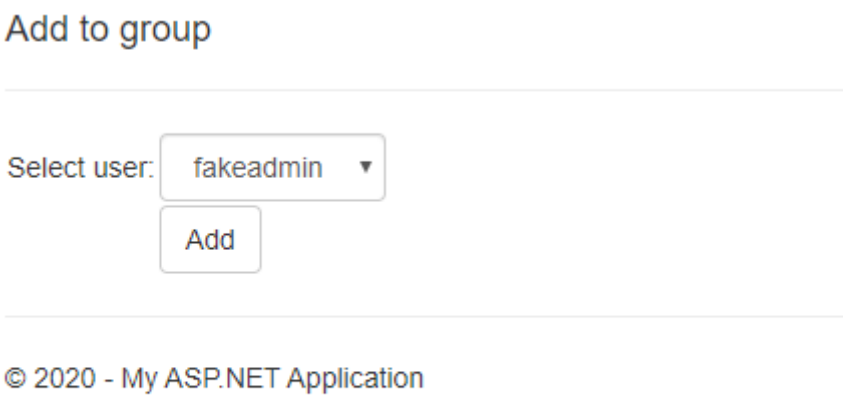
Ukoliko se želi pristupiti sekciji grupa za pojedini repozitorij, odabire se akcija *Groups* na stranici detalja za repozitorij. Na slici 3.18. prikazan je sadržaj stranice grupa.



Slika 3.18. Prikaz stranice repozitorija

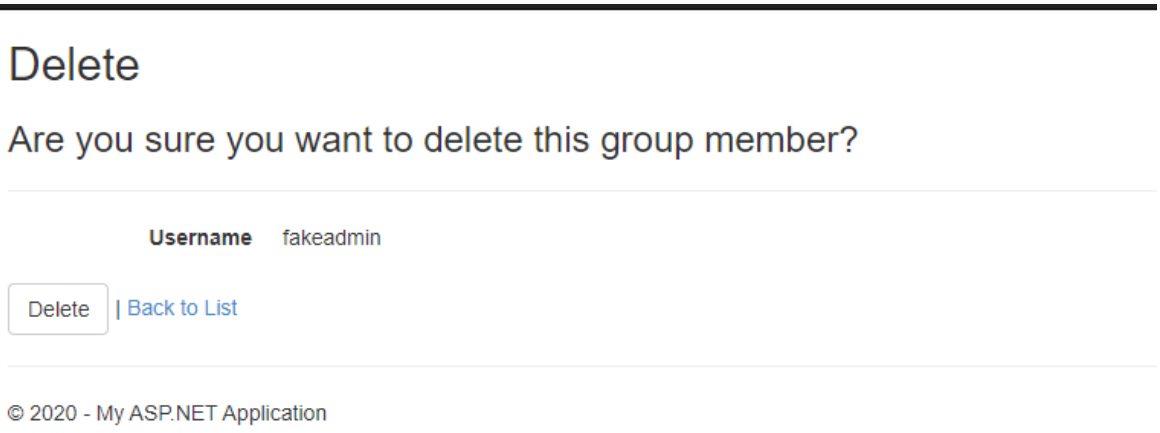
Stranica se sastoji od tabličnog prikaza korisnika koji su uključeni u repozitorij, te mogućnosti dvije akcije: dodavanja nove osobe u repozitorij i brisanje osobe iz repozitorija.

Pritiskom na akciju dodavanja (engl. *add*) otvara se novi prozor u kojem se odabire osoba koja se želi dodati iz padajućeg izbornika. Nakon odabira potrebno je pritisnuti gumb dodavanja kako bi se korisnik uspješno spremio. Na slici 3.19. prikazana je stranica dodavanja osobe u grupu.



Slika 3.19. Stranica dodavanja novog korisnika u grupu

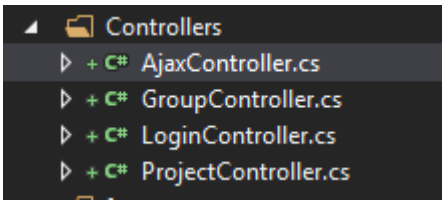
Pritiskom na akciju brisanja (engl. *delete*) nakon dodatne potvrde briše se odabrani korisnik iz repozitorija.



Slika 3.20. Prozor za potvrdu akcije brisanja osobe iz grupe

3.2.2. Implementacija funkcionalnosti

Web dio sustava sastoji se od 4 upravljača (engl. *controller*) koji vode preusmjeravanje korisnika unutar web stranice.



Slika 3.21. Upravljači web stranice

Svaki zahtjev koji dolazi do stranice proći će kroz jedan od upravljača, ovisno o željenom odredištu.

Pri prvom ulasku u stranicu osoba će biti preusmjerena na login upravljač koji će potom vratiti korisniku pogled (engl. *view*) u kojem će morati unijeti svoje podatke za prijavu.

```
[HttpGet]
public ActionResult Login()
{
      return View();
}
```

```
[HttpPost]
public ActionResult Login(string email, string password)
{
      UserAccountService service = new UserAccountService();
      ClaimsIdentity userIdentity = service.ValidateUser(email, password)
      as ClaimsIdentity;
      if(userIdentity != null)
      {
            Session["user"] = userIdentity;
            return RedirectToAction("Index", "Project");
      }
      return View();
}
```

Kod 3.14. Upravljač za logiranje

Nakon uspješne prijave osoba se preusmjerava na početnu stranicu upravljača projekta.

Na stranici se prikazuje lista repozitorija u kojima je uključen korisnik.

Akcija za uređivanje dohvaća repozitorij iz baze podataka i prikazuje stranicu za uređivanje naziva repozitorija korisniku.

```
public ActionResult Edit(int id)
{
      RepositoryService repositoryService = new RepositoryService();
      Repository userRepository =
      repositoryService.GetRepositoryById(id);
      return View(userRepository);
}
```

Kod 3.15. Metoda za uređivanje repozitorija na upravljaču

U kodu broj 3.15. se može vidjeti da metode upravljača vraćaju nazad poglede – njima se dodatno može proslijediti model kao parametar. Nakon toga podatci iz modela se šalju u datoteku nastavka *.cshtml* koja se mora nalaziti u mapi istog naziva kao upravljač. Nakon što se pogled ispuni podacima sadržaj se vraća korisniku. U kodu broj 3.16. prikazuje se isječak iz *.cshtml* datoteke.

```
@using (Html.BeginForm())
{
```

```
        @Html.AntiForgeryToken()
        <div class="form-horizontal">
                <h4>Repository</h4>
                <hr />
                        @Html.ValidationSummary(true, "", new { @class = "text-
                        danger" })
                        @Html.HiddenFor(model => model.IDRepository)
                <div class="form-group">
                        @Html.LabelFor(model => model.Name, htmlAttributes: new
                        { @class = "control-label col-md-2" })
                <div class="col-md-10">
                        @Html.EditorFor(model => model.Name, new {
                        htmlAttributes = new { @class = "form-control" } })
                        @Html.ValidationMessageFor(model => model.Name, "", new
                        { @class = "text-danger" })
                </div>
        </div>

        <div class="form-group">
                <div class="col-md-offset-2 col-md-10">
                        <input type="submit" value="Save" class="btn btn-
                        default" />
                </div>
        </div>
</div>
}
```

Kod 3.16. Isječak iz .cshtml pogleda

Nakon što korisnik pritisne akciju spremanja, preusmjeri ga se na isti usmjerivač, ali na metodu koja u sebi ima označen atribut `[HttpPost]`.

Unutar te metode kao parametar postavlja se model ažuriranih vrijednosti s kojima se onda ažurira stanje u bazi podataka.

Akcijska metoda detalja kao parametar prima identifikator repozitorija, dohvaća repozitorij iz servisa repozitorija te potom dohvaća listu projektnih datoteka iz Google Cloud skladišta datoteka.

Nakon toga vrijednosti prosljeđuje u pogled gdje se generira prikaz strukture projekta.

```
public ActionResult Details(int id)
```

```
{
        ProjectViewModel viewModel = new ProjectViewModel();
        RepositoryService repositoryService = new RepositoryService();
        Repository repository = repositoryService.GetRepositoryById(id);
        DownloadService downloadService = new DownloadService();
        IEnumerable<Project> projects =
        downloadService.GetProjectsById(repositoryService.GetRepositoryById
        (id), "").OrderByDescending(project => project.FileVersion);
        SelectList versionList = new SelectList(projects, "FileVersion",
        "FileVersion");
        ViewBag.VersionList = versionList;
        viewModel.IDRepository = id;
        return View(viewModel);
}
```

Kod 3.17. Metoda za prikaz detalja repozitorija

Kada korisnik pritisne na akciju prikaza grupa u pogledu detalja repozitorija, preusmjerava ga se na usmjerivač grupa koji vrši programsku logiku vezanu uz postavljanje i brisanje korisnika iz grupa.

3.2.3. Sigurnost podataka korisnika

Važno je osigurati privatnost projekta koji se nalaze u repozitorijima od korisnika.

Da bi se to osiguralo, sustav radi na sljedeći način:

- Server sadržava i upravlja projektnim datotekama svih korisnika.
- Korisnik dobiva povratne informacije sadržaja projekta kroz vizualni prikaz.
- Krajnji korisnik nema direktan pristup datotekama putem web aplikacije.

Kada korisnik zatraži detalje projekta putem akcije, sustav će u pozadini obaviti povezivanje s Google Cloud skladištem datoteka i izvršiti preuzimanje, učitavanje i vraćanje podataka u upravljač.

Nakon obrade tih podataka korisnik dobiva strukturu projekta i njegova ostala svojstva bez direktnog pristupa samoj datoteci.

3.2.4. Vizualizacija podataka

Nakon što se uspješno pripreme podatci projekta i pošalju se u pogled za prikaz detalja repozitorija, pogled obrađuje i vraća korisniku povratne informacije projekta.

Unutar pogleda nalazi se padajući izbornik u kojem se odabire željena verzija projekta za generiranje. Nakon što se odabere verzija, sav *HTML* sadržaj generira se putem *AJAX* poziva u koji se prosljeđuje identifikacijski broj projekta i identifikacijski broj verzije koja je odabrana.

Podatci se prosljeđuju na *AJAX* upravljač koji dohvaća odabranu verziju projekta i kreira parcijalni pogled (engl. *partial view*) unutar kojega se vrši programska logika generiranja *HTML* sadržaja. *AJAX* poziv kao rezultat dobiva sadržaj projekta u *HTML* obliku koji se dodaje na pogled detalja repozitorija.

```
public ActionResult GenerateProjectStructure(int idProject, int
idVersion)
{
      ProjectViewModel viewModel = new ProjectViewModel();
      RepositoryService repositoryService = new RepositoryService();
      DownloadService downloadService = new DownloadService();Project
      project =
      downloadService.GetProjectsById(repositoryService.GetRepositoryById
      (idProject), "").FirstOrDefault(proj => proj.FileVersion ==
      idVersion.ToString());
      ProjectViewModel model = new ProjectViewModel();
      model.Project = project;
      PartialViewResult result = PartialView("_ProjectStructure", model);
      return result;
}
```

Kod 3.18. Metoda za pripremu podataka za generiranje

Unutar parcijalnog pogleda iterira se kroz sadržaj poslanog projekta pomoću pokazivača (engl. *cursor*) te se programskom logikom računa distanca između objekta unutar projekta za svaku traku.

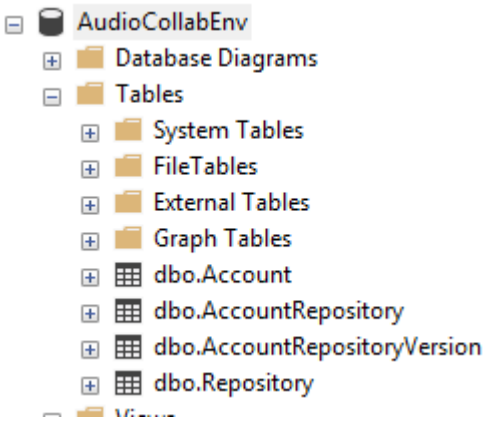
```
@foreach (var track in Model.Project.Tracks)
{
      <div class="project-track">
            <div class="left-track-display">
```

```
            @track.Name
            </div>
            <div class="right-track-display">
                  @{int cursor = 0;
                  foreach (var item in track.Items)
                  {
                        <div class="item-blank-space"
                        style="width:@(item.Position-
                        cursor)px;height:inherit">
                        </div>
                        cursor += item.Position - cursor;
                        <div class="item"
                        style="width:@(item.Length)px;height:inherit">
                        </div>
                        cursor += item.Length;
                        }
                  }
            </div>
            </div>
}
```

Kod 3.19. Generiranje sadržaja projekta

3.3. Baza podataka

Podatci se pohranjuju u *SQL Server* bazu podataka. Osim repozitorija, u *SQL Server* pohranjuju se korisnički računi i veze između korisničkih računa i repozitorija.

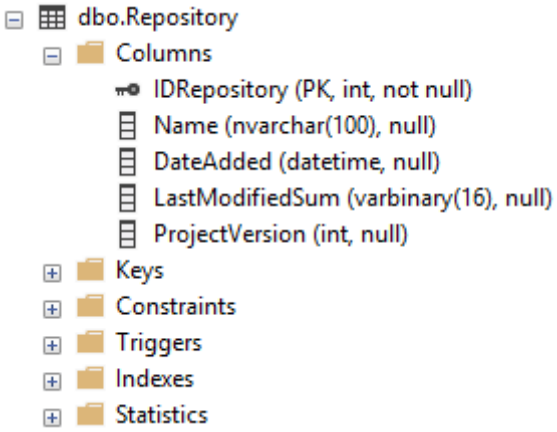


Slika 3.22. Struktura tablica baze podataka

Na slici 3.22. vidljiva je struktura baze podataka:

- Tablica *Account* sadrži podatke potrebne za prijave korisnika
- Tablica *AccountRepository* sadrži podatke za grupe korisnika
- Tablica *AccountRepositoryVersion* sadržava broj posljednje ažurirane verzije pojedinog korisnika u pojedinom repozitoriju
- Tablica *Repository* sadrži podatke o repozitorijima

3.3.1. Pohrana repozitorija

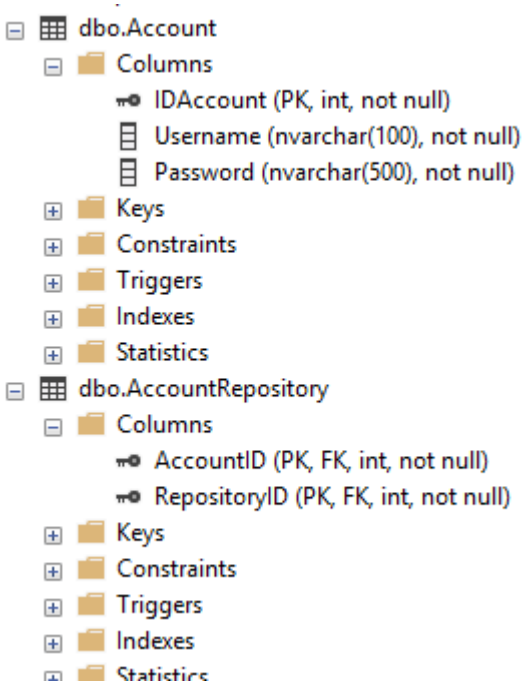


Slika 3.23. Struktura tablice repozitorija

Tablica repozitorija sadrži meta podatke uvezi repozitorija:

- Naziv repozitorija
- Datum dodavanja
- Koji je zadnji potpis datoteke koja je u bazi – ovaj podatak se koristi za provjeru prije sinkronizacije, da se utvrdi postoji li razlika između projekta koji se želi postaviti i trenutne posljednje verzije projekta
- Zadnja verzija projekta

3.3.2. Pohrana korisničkih računa



Slika 3.24. Struktura tablice računa korisnika

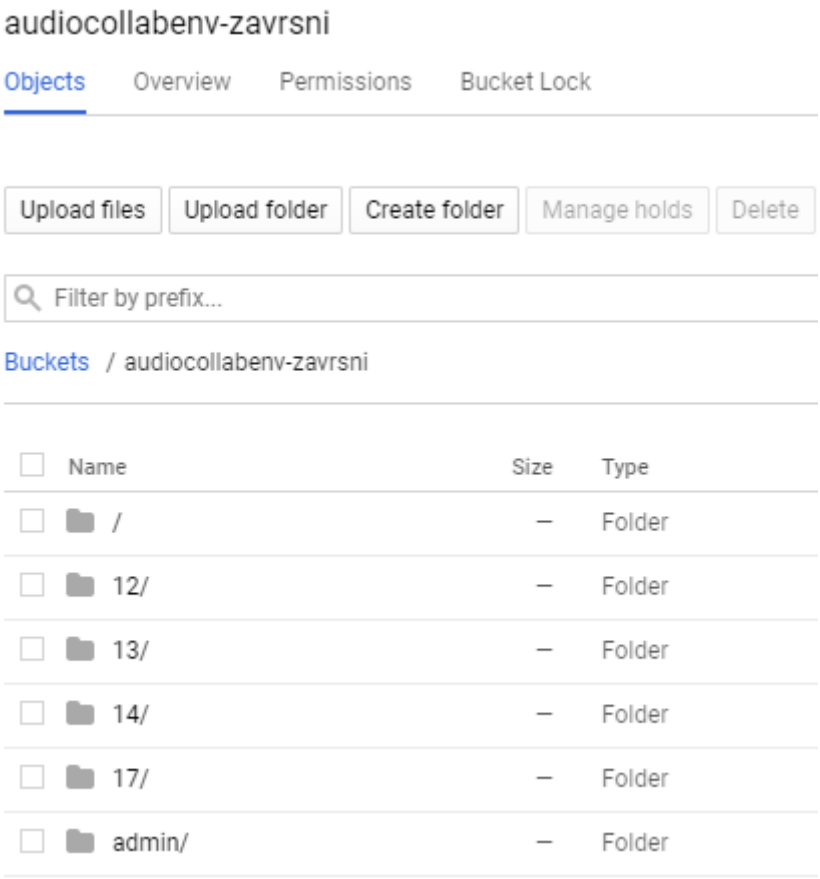
Podaci računa se pohranjuju u osnovnom obliku:

- Korisničko ime
- Lozinka

Kada se dodaje nova osoba u grupu za pojedini repozitorij, potrebno ju je dodati u tablicu *AccountRepository* koja je vidljiva na slici broj 3.24.

3.4. Skladište datoteka

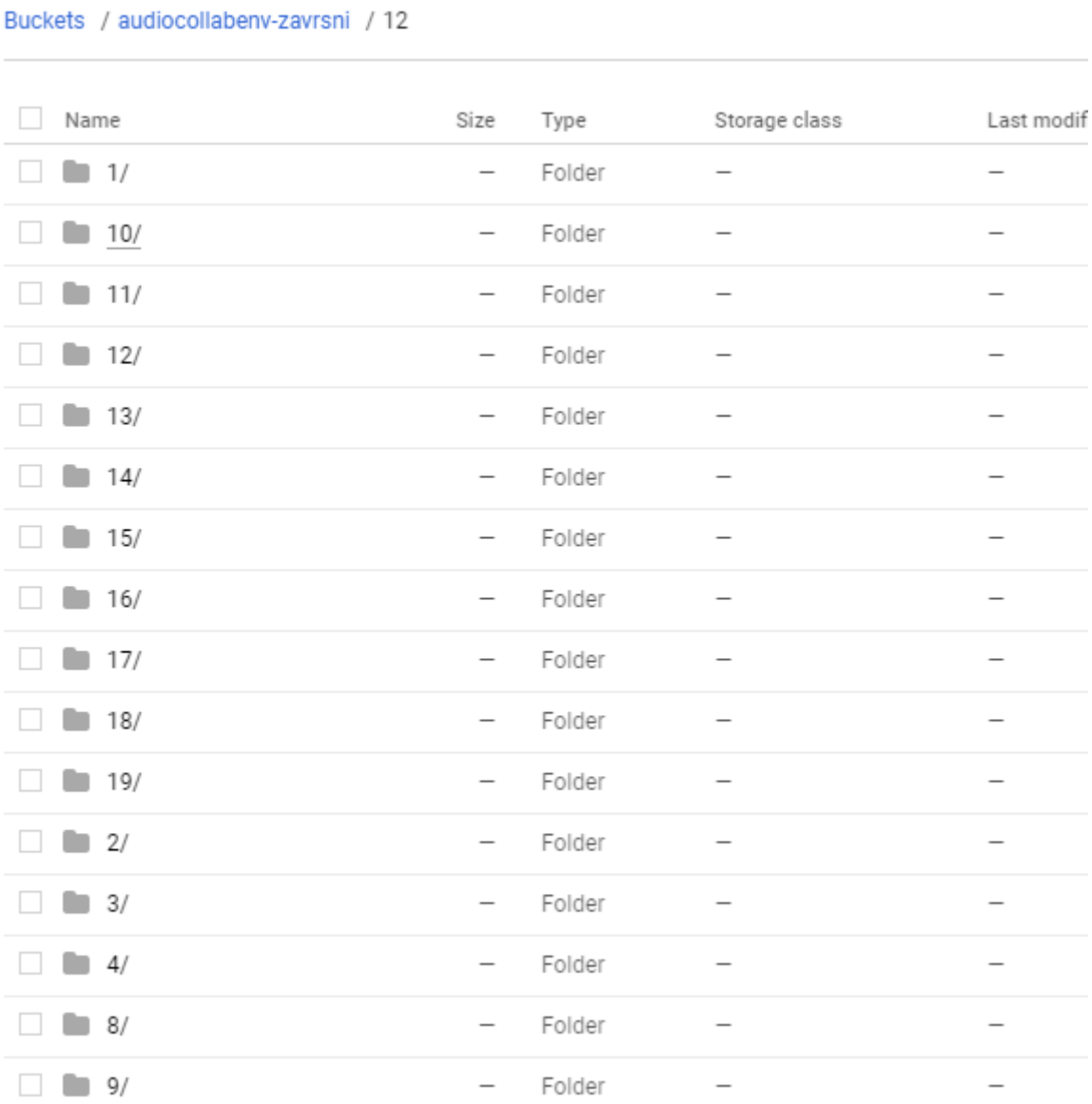
Pohrana datoteka vrši se u Google Cloud skladište podataka. Podatci se u Google Cloudu spremaju u kante (engl. *bucket*). Sve projektne datoteke nalaze se unutar jedne kante.



Slika 3.25. Grafički prikaz pohrane u *Google Cloudu*

Prilikom prve sinkronizacije novo nastalog repozitorija, kreira se mapa naziva identifikatora repozitorija u SQL Server bazi podataka. Na ovaj način osigurava se povezivanje SQL Servera i Google Cloud baze podataka.

Unutar mape pojedinog projekta nalazi se svaka verzija projekta kroz povijest. Prikaz mape projekta prikazan je na slici 3.26.



Slika 3.26. Prikaz mape projekta

3.5. Primjer korištenja

Za početak potrebno je kreirati novi repozitorij pomoću desktop aplikacije.

Nakon što je repozitorij kreiran, automatski se kreira odgovarajuća mapa u direktorij „*D:\CollabProjects*“.

Sada se može početi izrada projekta unutar FL Studio *software-a*. Nakon što je korisnik završio s radom, svoj projekt sprema u mapu repozitorija.

Potrebno je ubačenu projektnu datoteku sinkronizirati sa serverom. To se može učiniti pritiskom na gumb *Sync* u desktop aplikaciji. Nakon pritiska gumba pokrenut će se proces inicijalizacije koji će pronaći projektnu datoteku u mapi repozitorija, te početi sinkronizirati repozitorij sa serverom.

Ukoliko je sve prošlo dobro, repozitorij će biti označen zelene boje što znači da je repozitorij ažuriran na posljednju verziju projekta.

Korisnik nakon toga može otići na web aplikaciju, ulogirati se, pronaći svoj projekt i uključiti drugog korisnika u svoj projekt. Nakon što je drugi korisnik uključen, pri sljedećoj prijavi u desktop aplikaciji prikazat će mu se novi projekt u listi koji je označen žute boje i predstavlja potrebnu sinkronizaciju.

Drugi korisnik nakon što sinkronizira projekt dobiva posljednju verziju projekta, u ovome slučaju inicijalnu verziju od prvog korisnika, dobiva projektnu datoteku u svojoj mapi na kojoj može početi raditi i vršiti promjene.

Zaključak

Sustavi za verzioniranje su važni element u procesu izrade te je potrebno kvalitetno vođenje i planiranje da bi se postigao zadovoljavajući rezultat.

Kolaboracijski sustavi još nisu potpuno razvijeni u audio produkcijskom svijetu. Trenutno se razvijaju kao vanjski dodaci u produkciji, umjesto integracije u postojeće sustave.

U velikoj većini sustava također postoji desktop aplikacija koja vrši sinkronizaciju i web aplikacija za pregled strukture projekta.

U ovome radu nastojao se postići drugačiji pristup problematici kolaboracijskog sustava unutar svijeta audio produkcije. Završni proizvod je sustav koji obuhvaća desktop i web dio te ih objedinjuje u kompletno kolaboracijsko okruženje.

Kako će se tehnologija i dalje nastaviti razvijati u brzom toku, proširivati će se mogućnosti kolaboracijskih alata i njihovih integracija u postojeće sustave za izradu audio sadržaja.

Popis kratica

AJAX	*Asynchronous JavaScript + XML*	asinhroni javascript i xml
FLP	*FL Studio project*	projekt iz programa FL studio
HTML	*HyperText Markup Language*	prezentacijski jezik za izradu web stranica
MVC	*Model View Controller*	model pogled upravljač
SQL	*Structured Query Language*	strukturirani jezik upita
WPF	*Windows Presentation Foundation*	zaklada prezentacije sustava Windows
XAML	*Extensible Application Markup Language*	proširenI jezik označavanja aplikacije

Popis slika

Popis tablica

Popis kôdova

Literatura

[1] https://hr.wikipedia.org/wiki/Verzioniranje_softvera – 10.01.2020

[2] https://www.lynda.com/ALMTFS-tutorials/history-version-control/106788/115979-4.html - 13.01.2020

[3] https://pcchip.hr/softver/posao-i-financije/sustavi-za-upravljanje-verzijama-dokumenata-i-projekata/ - 10.01.2020

Prilog

ALGEBRA
VISOKO
UČILIŠTE

Sustav za kontrole verzija u kolaboracijskom okruženju

Pristupnik: Albert Huzejrović, 0321006184

Mentor: prof. dr. sc. Aleksander Radovan