Interaktivni online kviz

Ćuić, Filip

Master's thesis / Diplomski rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:462908

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-04-19**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

INTERAKTIVNI ONLINE KVIZ

Diplomski rad

Filip Ćuić

Osijek, 2020.

Sadržaj

1. UVOD 1
1.1 Zadatak diplomskog rada 2
2. PREGLED POSTOJEĆIH RJEŠENJA 3
2.1 PuzzGrid 3
2.2 ClassTools 4
3. ALATI I TEHNOLOGIJE 6
3.1 Microsoft Visio 6
3.2 Microsoft SQL Server Management Studio 7
3.2.1 SQL 8
3.3 ASP.NET 11
3.3.1 MVC programerski obrazac 12
3.3.2 Entity Framework 13
3.3.3 C# programski jezik 15
3.3.4 ASP.NET Web API 16
3.4 Postman 17
3.5 Angular 2+ 18
3.5.1 HTML 20
3.5.2 CSS 21
3.5.3 TypeScript 22
3.5.4 NGX Bootstrap 22
3.6 Visual Studio Code 23
4. IZRADA APLIKACIJE 25
4.1 Izrada relacijskog modela podataka 25
4.1.1 Zadani problem 25
4.1.2 Utvrđivanje i analiza normaliziranog modela podataka (ER modela) „Zida za spajanje“ 25
4.1.3 Pretvaranje ER modela u relacijski model podataka 26
4.2 Izrada poslužiteljske aplikacije u .NET okruženju 29
4.2.1 Spajanje aplikacije s bazom podataka 29
4.2.2 Model podataka u .NET okruženju 30
4.2.3 Definiranje upravljača 34

4.2.4 Korisnička autentikacija 36
4.2.5 Registracija korisnika po ulozi 39
4.2.6 Brisanje objekata u bazi 41
4.3 Izrada korisničkog sučelja 43
4.3.1 Podjela u komponente 43
4.3.2 Registracija i prijava korisnika 46
4.3.3 Restrikcija pristupa Admin Panel komponenti 49
4.3.4 Unos „Zida za spajanje“ 51
4.3.5 Dohvaćanje, prikaz i brisanje podataka 54
4.3.6 Otvaranje skočnih prozora, ažuriranje zida 59
4.3.7 Implementacija igranja kviza 61
5. ZAKLJUČAK 72
LITERATURA 73
SAŽETAK 75
ABSTRACT 76
ŽIVOTOPIS 77
PRILOZI 78

1. UVOD

Cilj diplomskog rada je izraditi aplikaciju koja će omogućiti korisnicima igranje i izradu interaktivnog online kviza. Ovakav tip kviza naziva se „Zid za spajanje“ (*eng. Connecting wall*). Glavni cilj rada je modeliranje baze podataka, izrada poslovne logike aplikacije, izrada korisničkog sučelja, te osiguravanje njihovog ispravnog međudjelovanja. Korisniku će biti omogućeno kreiranje vlastitog zida, izmjena kreiranog zida, kao i njegovo brisanje. Korisnici će se morati registrirati i prijaviti za kreiranje zida i igranje kvizova. Relacijska baza koja se koristi kod izrade aplikacije je Microsoft SQL baza podataka, a program koji omogućava upravljanje i pregled baze podataka je Microsoft SQL Server Management Studio. Za izradu poslovne logike aplikacije koristi se Microsoft .NET Framework Web Aplikacijsko programsko sučelje (*eng. Web Application Programming interface*). Kod modeliranja korisničkog sučelja koristit će se Angular 2+ okruženje za razvoj web aplikacija (Verzija 9).

Pri modeliranju i korištenju baze podataka (MSSQL) nužno je poznavanje SQL jezika, teorijsko poznavanje modeliranja podataka, modela entitet-veza, te objektnog modela podataka. Navedena znanja stečena su na kolegiju „Baze podataka“. Za izradu poslovne logike u programskom jeziku C# (C Sharp), nužno je poznavanje osnovne programske razvojne arhitekture MVC (*eng. Model-View-Controller*), objektno-orijentiranih načela, te strukture i funkcionalnosti jezika C#. Navedena su znanja stečena na kolegijima „Objektno-orijentirano programiranje“, „Računarstvo usluga i analiza podataka“, te „Stručna praksa iz računarstva“. Kod izrade korisničkog sučelja u Angular 2+ okruženju koristilo se znanje iz kolegija „Vizualizacija podataka“, „Web programiranje“,“Napredno web programiranje“, te „Stručna praksa iz računarstva“.

Izrada diplomskog rada dijeli se u 3 dijela :

1) Definiranje ER (eng. Entity-Relationship) modela problema, te izrada modela preko C# klasa (Code First pristup). Korištenjem ASP.NET migracija C# klase će se modelirati u Microsoft SQL-u. Model podataka će zbog lakše vizualizacije također biti prikazan u programu Microsoft Visio.
2) Izrada poslovne logike problema u programskom jeziku C#. To podrazumijeva kreiranje kontrolera i organizaciju podataka koji će se slati i primati s korisničkog sučelja.

3) Izrada korisničkog sučelja u Angular 2+ okruženju. Odvajanje web aplikacije u više zasebnih komponenti koje funkcioniraju odvojeno. Također će se izvršiti modeliranje zahtjeva s klijentske strane.

U drugom poglavlju objasnit će se slična ostvarenja ovom diplomskom radu (*eng. State of Art*). Treće poglavlje objasnit će alate i tehnologije koji su se koristili pri izradi rada, a u četvrtom poglavlju se opisuje detaljan postupak izrade rada. U petom poglavlju će se donijeti zaključak.

1.1 Zadatak diplomskog rada

Zadatak diplomskog rada je izraditi aplikaciju koja omogućuje igranje kviza „Zid za spajanje“, te korisničku izradu vlastitih zidova za spajanje. Igra funkcionira tako da jedan zid ima 16 pojmova raspoređenih u četiri reda i stupca. Aplikacija će nasumično izmiješati pojmove, a igrač mora grupirati 4 pojma koja su povezana po nekom skrivenom kriteriju. Nakon uspješnog grupiranja, igrač ima priliku ostvariti dodatne bodove tako što objašnjava povezanosti pojmova. Osim igranja kvizova, prijavljeni korisnici moći će napraviti svoj zid tako što će upisati četiri pojma koja su povezani, te odrediti prihvatljivi termin po kojima su navedeni pojmovi vezani. Korisnicima će također biti omogućena izmjena vlastitih zidova (pojmova i veza između pojmova). Igra je inspirirana BBC-ovim kvizom Only Connect.

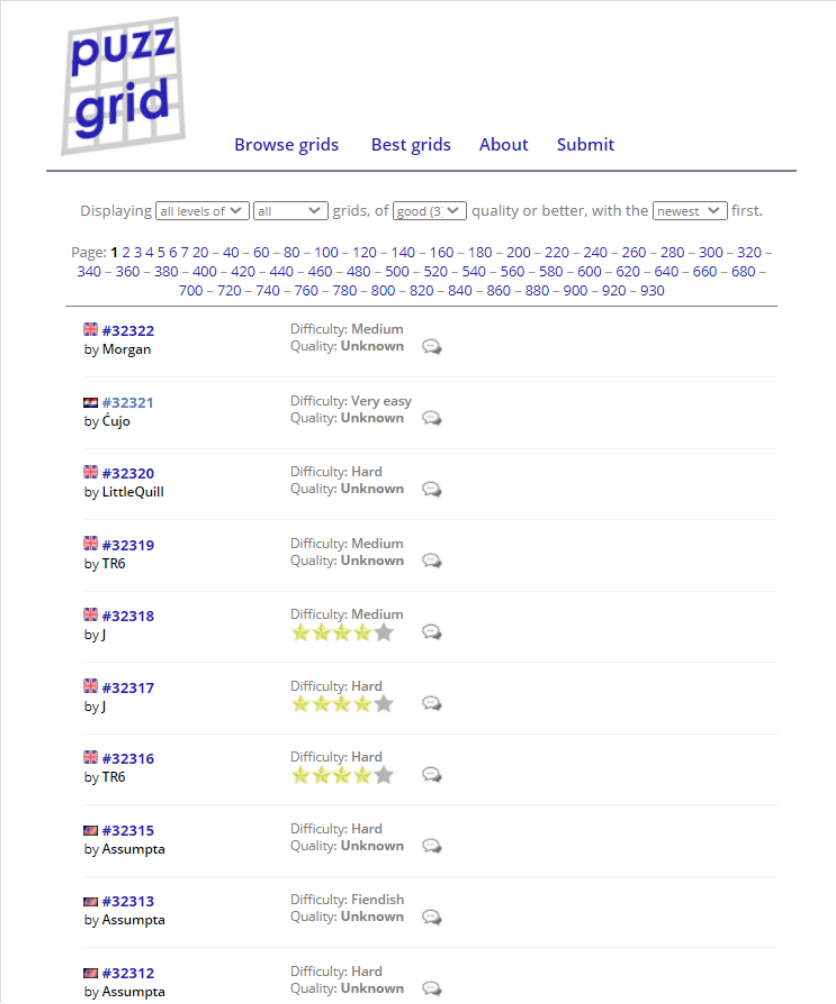
Slika 1.1. *Primjer zida za spajanje s predloženim rješenjima.[17]*

2. PREGLED POSTOJEĆIH RJEŠENJA

U ovom poglavlju opisat će se slična ostvarenja ovom radu, odnosno postojeće online aplikacije koje omogućuju rješavanje i kreiranje igre zid za spajanje (*eng. connecting wall*). Trenutno postoje dvije slične online aplikacije : PuzzGrid i ClassTools. Pregled svake aplikacije pojedinačno bit će obavljen u sljedećim potpoglavljima.

2.1 PuzzGrid

PuzzGrid je web stranica koja omogućuje igranje igre bazirane na BBC-evom kvizu Only Connect. Ljudima omogućuje zabavu kroz kreiranje vlastitih zidova i njihovo dijeljenje. Igranje kviza i njegovo kreiranje je besplatno, no ljudima koji podržavaju njihov rad plaćanjem omogućuju i korištenje stranice bez reklama (ad-free access), te stavljanje slika kao pojmove u zidove koje kreiraju. Kod stvaranja kviza korisnici navode težinu kviza, svoju nacionalnost (omogućuje klasifikaciju kvizova), ime autora kviza (kao pseudonim ili kao ime), te pojmove za svaku od 4 grupe i neke od prihvatljivih izraza kojima se može opisati povezanost pojmova. Na kraju, korisnici mogu unijeti svoju E-mail adresu (proizvoljno) i odrediti je li njihov zid prikladan za rješavanje široj publici. Kod rješavanja kvizova koje su napravili drugi korisnici, korisnik može filtrirati prikaz kvizova po težini, geografskom području za koje je kviz namijenjen (Ujedinjeno Kraljevstvo, Sjedinjene Države, Kina ili Njemačka), kvaliteti napravljenog kviza (određuju je korisnici koji odigraju kviz) i poredati ih po datumu kreiranja. Na stranici postoji oko 20 tisuća napravljenih kvizova.

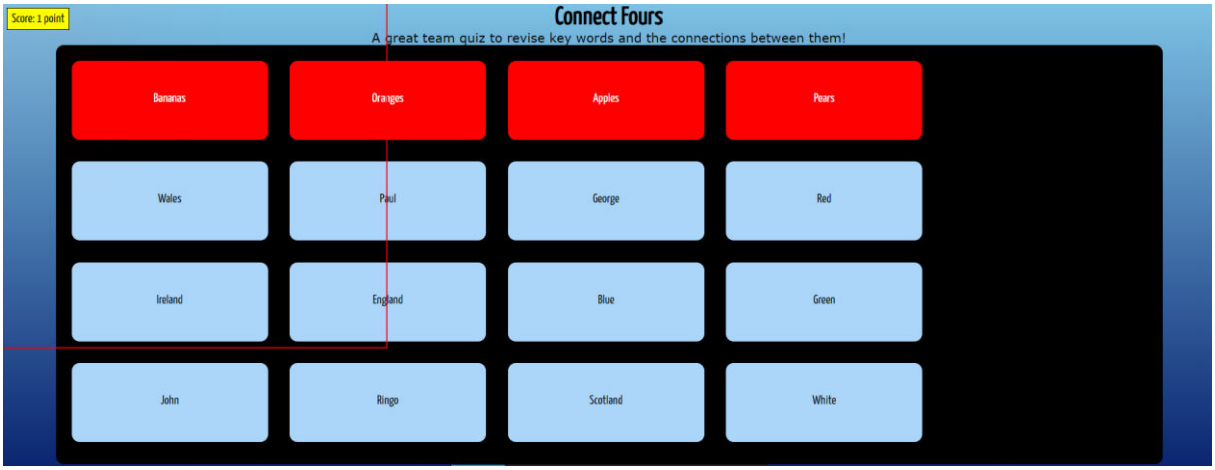


Slika 2.1. *Prikaz počenog sučelja web mjesta PuzzGrid.com.*

2.2 ClassTools

ClassTools je web stranica koja omogućuje kreiranje besplatnih igara, kvizova, aktivnosti i dijagrama, te njihovo dijeljenje i rješavanje. Postoje razni predlošci za kvizove i igre koje se mogu kreirati i igrati (primjerice : igre sortiranja, pitalice s ponuđenim odgovorima, generator arkadnih igara itd.). Jedan od predložaka kviza koji se može igrati i kreirati je upravo zid za spajanje, a na ovoj stranici predložak je nazvan Connect Fours. Za razliku od PuzzGrid-a, kreiranje i igranje predloška Connect Fours je dostupno samo korisnicima s plaćenim članstvom (*eng. Premium Account*). Kao primjer, svim ostalim korisnicima omogućeno je igranje samo jednog kviza, to jest primjera kviza.

Slika 2.2. *Prikaz početnog sučelja predloška za kviz Connect Fours na ClassTools.net web stranici.*

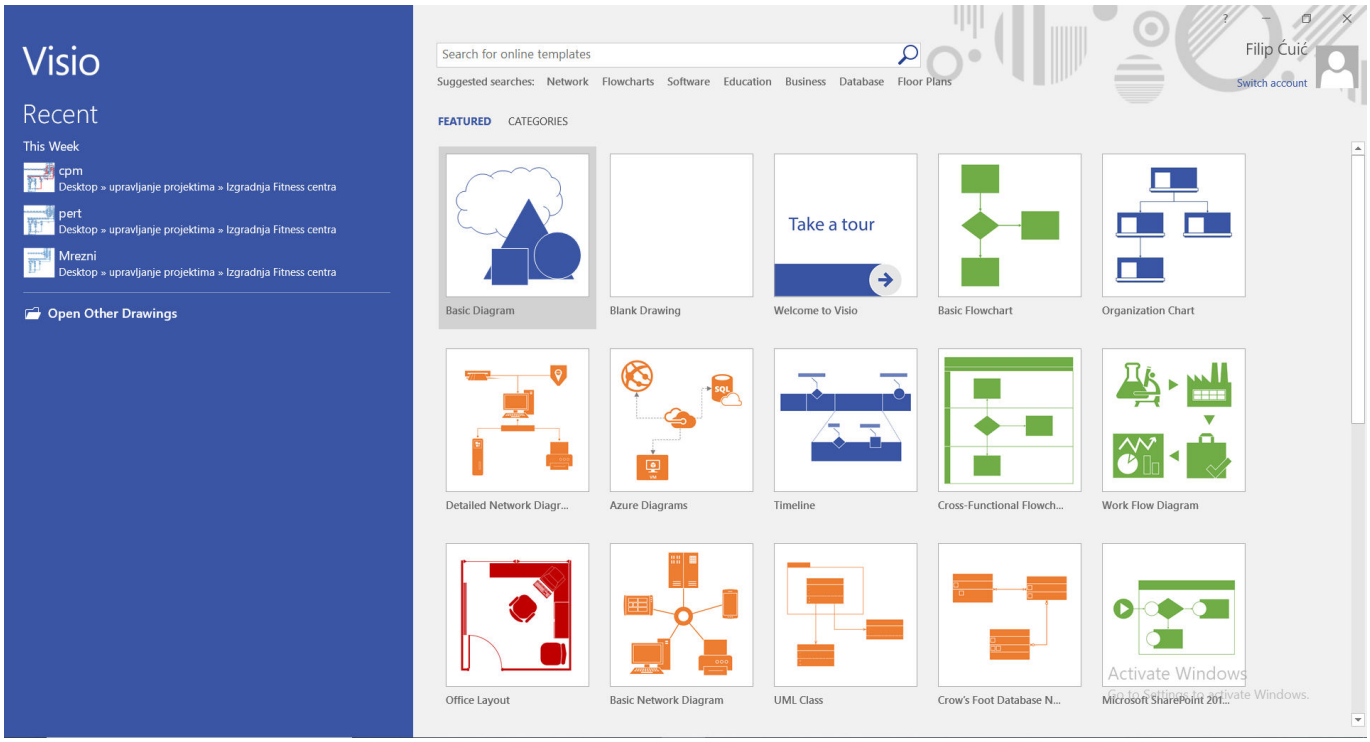


Slika 2.3. *Primjer kviza Connect Fours na ClassTools.net web stranici.*

3. ALATI I TEHNOLOGIJE

3.1 Microsoft Visio

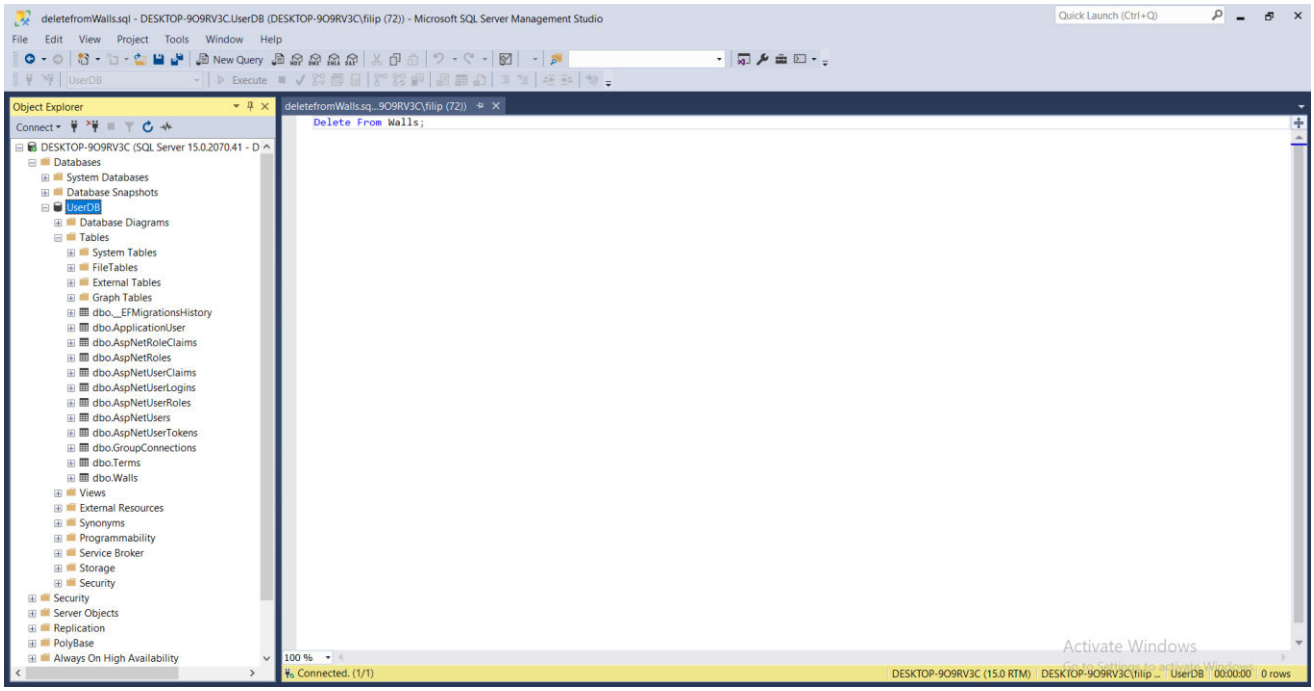
Microsoft Visio je alat za razvoj dvodimenzionalnih grafičkih elemenata i dijagrama. Prva verzija je razvijena 1992. godine, a razvila ju je tvrtka Shapeware Corporation. Od 2000. godine je dio Microsoft Office programskog paketa. Do danas je razvijeno 16 verzija aplikacije, a posljednja verzija razvijena je 2019. godine (v16.0). Trenutna verzija sadrži preko tisuću vrsta online predložaka podijeljenih u sedam osnovnih kategorija : mrežni dijagrami, dijagrami toka, programska podrška, obrazovanje, poslovanje, baza podataka i tlocrti. Visio uvelike olakšava prikaz i omogućava lakšu vizualizaciju u svim navedenim područjima rada. U ovom diplomskom radu Visio će se koristiti za prikaz Entity-Relationship modela podataka interaktivnog online kviza.



Slika 3.1. *Prikaz početnog sučelja programskog okruženja Microsoft Visio.*

3.2 Microsoft SQL Server Management Studio

SQL Server Management Studio (SSMS) je razvojna okolina za pristupanje, konfiguriranje, upravljanje, razvijanje i administraciju SQL servera i razvoj objekata baze podataka. Prva verzija razvijena je 2005. godine, a naslijedila je sličnu programsku podršku Enterprise Manager koja se koristila za SQL 2000 i prethodne verzije. Razvojna okolina omogućuje dobar prikaz objekata servera, kao i rad s objektima i značajkama servera preko skriptnih uređivača (eng. Script editor) i grafičkih alata. Kod pokretanja aplikacije SSMS-a potrebna je autentikacija korisnika koja se može obaviti kao Windows autentikacija, SQL autentikacija ili autentikacija preko Microsoft Azure okoline. Najvažnija značajka SSMS-a je pretraživač objekata (*eng. Object Explorer*) koji korisniku omogućuje pretraživanje, odabir i interakciju s objektima SQL servera. Pretraživač objekata također omogućuje pregled svih definiranih baza podataka (*eng. Databases*) s njenim tablicama i n-torkama, sigurnosnih stavki (*eng. Security*), objekata servera (*eng. Server Objects*), serverovih odgovora (*eng. Replication*) i mnogih drugih stavki. Od ostalih značajki SSMS-a valja spomenuti i SQL prostor za upite (*eng. SQL Query*) koji omogućuje definiranje i upravljanje relacijskim modelom podataka, što se može izvršiti i preko grafičkog sučelja (opcije pretraživača objekata). Posljednja izdana verzija SSMS-a je verzija 18.5. U ovom diplomskom radu SQL Server Management Studio koristit će se za pregled, definiranje i upravljanje relacijskim modelom podataka interaktivnog online kviza.



Slika 3.2. *Prikaz sučelja Microsoft SQL Server Management Studia 18.5.*

3.2.1 SQL

SQL (*eng. Structured Query Language*) je jezik korišten u programiranju, a konstruiran je za upravljanje podacima unutar sustava relacijske baze podataka (*eng. Relational Database Management System*). Pogodan je za rukovanje strukturiranim podacima, to jest podacima koji se sastoje od raznih entiteta i varijabli.

Najčešće upotrebe SQL jezika su :

1) Ažuriranje, stvaranje i upiti nad bazom podataka,
2) Brisanje, mijenjanje i dodavanje redova (atributa) i n-torki (objekata) relacijskog modela podataka u bazi
3) Kontrola pristupa bazi podataka i njezinim objektima

Glavna shema relacijske baze podataka je u obliku tablice, što znači da je svaki definirani objekt jedna instanca tablice, a njegovi atributi su redovi tablice. Svaka tablica (relacija) sastoji se od zaglavlja (skupa atributa) i vrijednosti pojedinog atributa. Vrijednosti stupaca tablice (relacije) predstavljaju skup svih atributa tablice, a vrijednosti redaka predstavljaju pojedinu n-torku (objekt, instancu) tablice. Primjerice, tablica (relacija) ZidZaSpajanje sastojat će se od zaglavlja koje obuhvaća imena atributa : wallID (jedinstveni identifikator), wallName (ime zida), dateCreated (datum kreiranja) i korisnički identifikator (strani ključ). Sve vrijednosti jednog retka tablice ZidZaSpajanje predstavljat će jednu instancu zida za spajanje (npr. wallID : D07D47E9-58CC-4661-8F32-8682FA2BB88E, wallName : PremierLeague, dateCreated : 2020-05-19 10:56:35.5199005, userID : f858b438-e392-4cba-8b9c-45704aa37068). Ovakav relacijski model bazira se na relacijskoj algebri i izračunu relacija (*eng. Tuple Relational Calculus*).

SQL se sastoji od više vrsti upita među kojima su :

1) **DDL upiti** (*eng. Data Definition Language*) – upiti za upravljanje/definiranje tablica, ključne riječi ALTER/CREATE/DROP/TRUNCATE TABLE,
2) **DML upiti** (*eng. Data Manipulation Language*) – upiti za ažuriranje, dohvaćanje, uklanjanje, unošenje i upravljanje n-torkama (objektima) baze (prethodno moraju biti definirani DDL upitima), ključne riječi DELETE/INSERT/MERGE/SELECT/UPDATE TABLE,

3) **DCL** (*eng. Data Control Language*) – upiti koji omogućuju kontrolu pristupa podacima u bazi, te osiguravaju sigurnost podataka, ključne riječi EXECUTE AS, GRANT, REVOKE clause/statement
4) **TCL** (*eng. Transaction Control Language*) – upiti za upravljanje transakcijama i promjenama uzrokovanim DML upitima u bazi podataka, omogućuju i grupiranje više upita u logičke transakcije (grupe upita), ključne riječi COMMIT, ROLLBACK, SAVEPOINT

SQL pruža brojne tipove podataka koji definiraju vrijednost koju sadržava svaki atribut definirane tablice (primjerice, ime korisnika zahtjeva znakovne vrijednosti). Tri osnovna tipa podataka su string (tekstualni tip), numeric (brojčani tip) i „date and time“ (vremenska oznaka).

Neki od tipova podataka koji spadaju u 3 osnovna tipa podataka :

1) **Tekstualni tipovi podataka :** CHAR, VARCHAR, BINARY, VARBINARY, TINYTEXT, TEXT, MEDIUMTEXT, LONGTEXT, SET, ENUM
2) **Brojčani tipovi podataka :** BIT, TINYINT, BOOL, BOOLEAN, SMALLINT, MEDIUMINT, INT, BIGINT, FLOAT, DOUBLE, DECIMAL
3) **Tipovi podataka za vremenske oznake :** DATE, DATETIME, TIMESTAMP, TIME, YEAR

Od ostalih tipova podataka važno je spomenuti sql_variant (spremanje raznih tipova podataka), xml (spremanje XML formatiranih podataka), cursor (spremanje referenci podataka u pokazivač) i uniqueidentifier (spremanje jedinstvenog identifikatora, GUIDa).

U SQL-u je moguće dodatno ograničiti domenu vrijednosti (skup vrijednosti) određenog atributa korištenjem SQL ograničenja (SQL Constraints). Ograničenja se mogu navesti kod kreiranja tablice (CREATE TABLE) ili njezinog mijenjanja (ALTER TABLE). Najčešće korištena ograničenja u SQL-u su :

1) NOT NULL – osigurava da atribut (stupac) relacije nikad nije prazan (bez vrijednosti)
2) UNIQUE – osigurava da se vrijednosti navedenog atributa razlikuju za svaku n-torku (npr. Svaki identifikator osobe u bazi bi trebao biti jedinstven)
3) PRIMARY KEY – primarni ključ tablice i atribut koji osigurava jedinstvenost objekta u tablici, po njemu se svaka n-torka tablice mora razlikovati, kombinacija NOT NULL i UNIQUE ograničenja

4) FOREIGN KEY – strani ključ tablice, atribut koji omogućuje referenciranje (vezu) na primarni ključ neke druge tablice i tako uspostavlja jedinstvenu vezu između objekata (n-torki) dviju tablica (relacija)
5) CHECK – osigurava da sve vrijednosti pojedinog atributa relacije imaju odgovarajući raspon (npr. Za tablicu punoljetnih osoba dobro je provjeravati je li atribut „godine“ veći ili jednak 18), omogućuje nabrajanje prihvatljivih vrijednosti (CHECK IN) kao i korištenje logičkih operatora (CHECK (godine>=18) AND (godine<65))
6) DEFAULT - postavlja predefiniranu vrijednost stupca ako ona pri unosu nije navedena
7) INDEX – osigurava brže dohvaćanje podataka iz baze

SQL pruža nekoliko desetaka gotovih funkcija za operacije nad svakim tipom podatka (tekstualni, brojčani ili vremenske oznake), kao i napredne funkcije (pretvorba tipa podataka, dohvaćanje trenutnog korisnika i sl.). Osnovna podjela funkcija u SQL-u je na :

1) **Skalarne funkcije** – vraćaju jednu izlaznu vrijednost koja je bazirana na jednoj ulaznoj vrijednosti, odnosno vrijednosti samo jednog atributa (Primjer : funkcija ROUND() zaokružuje broj na određeni broj decimala).
2) **Agregatne funkcije** – vraćaju jednu izlaznu vrijednost, ali djeluju nad skupom vrijednosti određenog atributa (Primjerice, ako postoji atribut ocjena u tablici studenti, prosječna ocjena svih studenata dobiva se upitom SELECT AVG (ocjena) FROM studenti).

```
Insert into Walls(wallName,userID) values ('premierleague','274eed93-fedf-4825-9c11-deb1e7678b99');
```

Slika 3.3. *Unošenje n-torke u napravljenu tablicu preko SQL upita.*

3.3 ASP.NET

ASP.NET je besplatno okruženje za razvoj web aplikacija koje se sastoji od alata, biblioteka i programskih jezika. Okruženje je razvio Microsoft kako bi programerima omogućio razvoj web aplikacija, dinamičkih web stranica i programskih servisa. Verzija 1.0 izdana je u Siječnju 2002. godine i naslijedila je ASP (*eng. Active Server Pages*), Microsoftovo prvo poslužiteljsko skriptno okruženje za dinamičku izradu web stranica. ASP.NET aplikacije izvršavaju se na strani poslužitelja, te generiraju sadržaj koji se može iskoristiti s klijentske strane (*eng. frontend*). ASP .NET također ima mogućnost upravljanja bazom podataka, čime omogućuje jednostavno kreiranje aplikacija Code-First pristupom (baza podataka može se kreirati pisanjem koda u nekom od podržanih .NET jezika, nije nužno njezino zasebno projektiranje). Entity Framework je okvir koji omogućuje mapiranje objekata pisanih u kodu u DBO (*eng. DataBase Object*), objekte u bazi podataka. Pošto pripada .NET platformi (razvojnom okruženju), podržava programiranje u svim jezicima podržanim u zajedničkoj jezičnoj okolini CLR (*eng. Common Language Runtime*) .NET platforme. Neki od jezika u kojima se mogu razvijati .NET aplikacije su : C# (C Sharp), Visual C++, Visual Basic i F#. ASP.NET osigurava niz programskih modela za stvaranje web aplikacija :

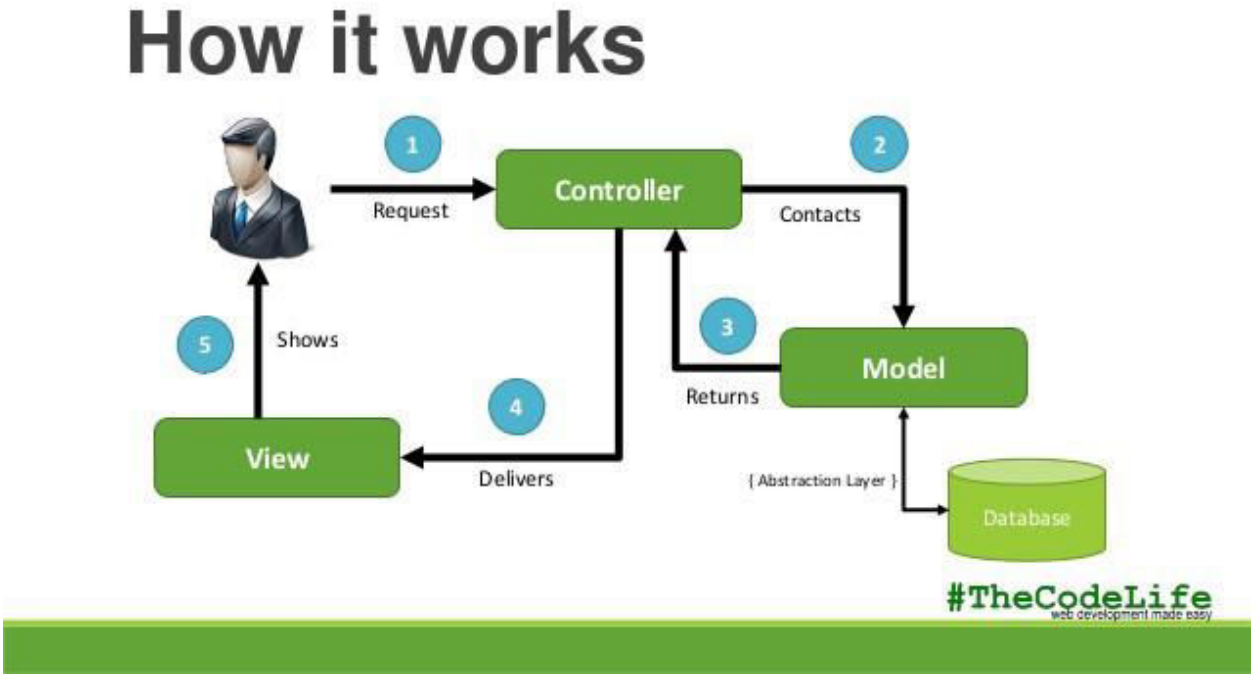
1) **ASP.NET Web forme** – model za izgradnju web stranica iz više odvojenih komponenti, gdje se elementi korisničkog sučelja procesiraju na poslužiteljskoj strani
2) **ASP.NET MVC** – omogućuje izgradnju web stranica koristeći Model-Pogled-Kontroler (*eng. Model-View-Controller*) obrazac za dizajniranje aplikacija
3) **ASP.NET Web Pages (ASP.NET Razor)** – model koji omogućuje paralelnu izgradnju poslužiteljske strane (najčešće u C# programskom jeziku) i sadržaja klijentske strane (pisana u jezicima HTML, CSS, JavaScript)
4) **ASP.NET Web API** – programski model koji omogućuje izgradnju aplikacijskog programskog sučelja (*eng. Application Programming Interface - API*) koje služi kao poslužitelj, komunikacija s klijentskom stranom vrši se preko HTTP zahtjeva koji se modeliraju po kontrolerima definiranog API-a
5) **ASP.NET Webhooks** – model koji se koristi za pretplaćivanje i objavljivanje događaja preko HTTP-a
6) **SignalR** – model koji omogućuje obostranu komunikaciju u stvarnom vremenu između klijenta i servera

3.3.1 MVC programerski obrazac

MVC (*eng. Model-View-Controller*) je najčešće korišteni obrazac za razvoj programske podrške. Koristi se kod razvoja korisničkih sučelja gdje je programska podrška podijeljena u 3 zasebne cjeline. Takva podjela omogućuje veću preglednost izvornog koda, te lakše testiranje i održavanje programske podrške. MVC je prvi put uveden 1979. godine u SmallTalk programski jezik, a uveo ga je Trygve Reenskaug. MVC pogodan je za aplikacije u kojima je potrebno odvojiti prikazane podatke korisniku od podataka koji mu omogućuju prikaz i rad u projektiranoj programskoj podršci. Dijelovi (komponente) MVC-a su:

1) **Model** – središnja komponenta obrasca. Model se odnosi na podatkovnu strukturu i poslovnu logiku aplikacije. Propisuje pravila aplikacije, njezinu logiku, te oblik podataka s kojom aplikacija „rukuje“. Napravljeni model vezan je za bazu podataka, to jest, baza podataka mora imati tablice i atribute projektirane po programskom modelu. Projektiranje modela omogućeno je u Entity-Framework okviru .NET okruženja, a njegovom kreiranju se može pristupiti na dva načina : Projektiranje koda prvo (*eng. Code First*) i projektiranje Baze podataka prvo (*eng. Database First*).
2) **Pogled** *(eng.View)* – predstavlja bilo kakav prikaz informacija (grafovi, dijagrami, tablice i sl.) na korisničkom sučelju, odnosi se na prezentaciju biranih podataka aplikacije na korisničkom sučelju. Pogledi se često oblikuju HTML (*eng. HyperText Markup Language*) dokumentima uz dodatno stilizirane elemente (CSS – *Cascading Style Sheets*) i rukovanje događajima stranica (JavaScript ili TypeScript). Alternativan način prikaza podataka aplikacije je korištenje okruženja za razvoj klijentske strane aplikacije (*eng. Frontend Framework*) kao što su : React.js, Vue.js, Angular itd. Ukratko, pogled predstavlja prezentaciju podataka napravljenog modela u određenom formatu.
3) **Kontroler/Upravljač** (*eng. Controller*) – komponenta obrasca koja prihvaća i obrađuje korisničke zahtjeve, vrši interakciju s podacima u modelu, te oblikuje pogled u skladu s obavljenom operacijom. Pisanjem kontrolera programer definira što će se izmijeniti u pogledu i modelu (bazi podataka) kod specifičnog unosa korisnika. Podaci se mogu unositi, dohvaćati i brisati pomoću HTTP metoda za REST (*eng. Representational State Transfer*) programsko aplikacijsko sučelje. Takva izmjena podataka najčešće sadrži podatke u JSON (*eng. JavaScript Object Notation*) formatu koji je pogodan za interakciju s modelom koji je također projektiran objektno-orijentiranom paradigmom. Ukratko,

pisanjem kontrolera programeri definiraju što im korisnici moraju poslati s klijentske strane kako bi im se vratili željeni podaci.

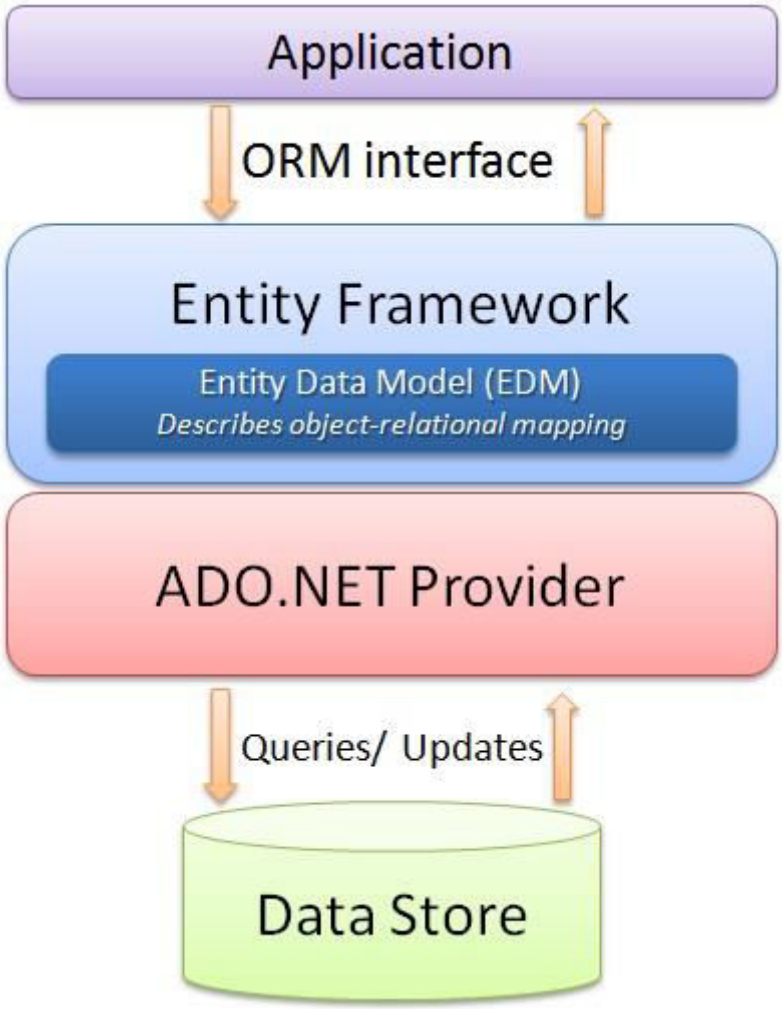


Slika 3.4. *Prikaz rada aplikacije projektirane po MVC obrascu. [18]*

3.3.2 Entity Framework

Entity Framework je besplatno razvojno okruženje za objektno-relacijsko mapiranje, a dio je ADO.NET (*eng. ActiveX Data Objects*) tehnologije pristupanja podacima. Prva verzija razvijena je 2008. godine kao dio .NET razvojnog okruženja , a najnovija verzija 6 se od 2020. godine razvija odvojeno od .NET-a. Entity Framework primjenjuje se u razvijanju podatkovno-orijentiranih aplikacija. Takve aplikacije imaju podatke organizirane kao objekte koji se sastoje od određenih atributa i metoda. Kako bi se objekti jednostavno i ispravno modelirali u bazi, potrebna je određena pretvorba objekata objektno-orijentiranog programskog jezika u objekte relacijske baze podataka (SQL). Pristup razvoja programske podrške koji radi na pretvorbi objekata baze podataka (*eng.* DBO – *DataBase Object*) u objekte u određenom programskom jeziku naziva se Database First pristup. Suprotni pristup naziva se Code First, kod njega se prvo kreiraju klase i objekti klasa u programskom jeziku koji se kasnije pomoću Entity Framework okruženja pretvoriti u objekte baze podataka. Od pristupa razvoja programske podrške potrebno

je još spomenuti Model First pristup kod kojeg se model podataka nekad kreirao u vizualnom dizajneru Entity Frameworka, a najnovija verzija EF više ne podržava ovakav pristup. Entity Framework omogućuje upravljanje objektima baze podataka bez pisanja upita u SQL-u, dovoljno je samo koristiti metode objekata u programskom jeziku kako bi se upravljalo bazom podataka.



Slika 3.5. *Prikaz arhitekture aplikacije razvijene u .NET okruženju s Entity Frameworkom kao posrednikom između objekata aplikacije i objakata baze podataka. [19]*

3.3.3 C# programski jezik

C# (C Sharp) je objektno-orijentirani programski jezik opće namjene. Dio je .NET platforme, a razvili su ga razvojni programeri Microsofta na čelu s Andersom Hejlsbergom. Sintaksa jezika slična je njegovim prethodnicima C-u, C++ i Visual Basicu, te programskim jezicima Javi i JavaScriptu. Neke od značajki sintakse C#-a su : završavanje naredbi/izjava (*eng. statement*) točka-zarezom (';') , grupiranje naredbi unutar petlje, funkcije ili grane programa pomoću kovrčastih zagrada('{}'), deklaracija varijabli znakom '=', uspoređivanje vrijednosti varijabli s '==', komentiranje pomoću '//' ili '/**/' itd. Pošto je objektno-orijentiran jezik, osnovna značajka mu je mogućnost organizacije varijabli i funkcija u atribute i metode klasa, što ga čini idealnim za razvoj podatkovno-orijentiranih aplikacija , te omogućuje jednostavnije projektiranje baze podataka. Od ostalih značajki objektno-orijentiranih jezika sadrži :

1) **Nasljeđivanje** – svaka projektirana klasa može naslijediti atribute i metode neke druge klase (Primjerice, klase Čovjek i Pas naslijedit će atribute i metode klase Sisavac, ali će im se atributi i metode svejedno međusobno razlikovati).
2) **Polimorfizam** – više klasa može naslijediti metode neke natklase, ali će se metode svake klase svejedno međusobno razlikovati (Primjerice, Klasa Zaposlenik ima metodu predstaviSe(), klasa Kuhar nasljeđuje klasu Zaposlenik, sadržava funkciju predstaviSe(), koja je različita od drugih klasa koje također nasljeđuju klasu Osoba kao : Konobar, ŠefSale itd.).
3) **Objekt** – instanca napravljene klase, sastoji se od atributa i metodi klase kojoj pripada (npr. Klasa Zaposlenik, instanca PeroPeric).
4) **Enkapsulacija** – objedinjavanje podataka (atributa) i metoda koje djeluju nad njima unutar svake klase (Primjerice, klasa BankovniRačun ima atribut (podatak) stanjeRačuna, kao i metodu provjeriStanje() koja vraća vrijednost atributa stanjeRačuna), također sadrži i modifikatore pristupa (private – varijabli se može pristupiti samo unutar klase; protected – metode klasa i potklase mogu pristupiti varijabli; public – svaka klasa može pristupiti varijabli).

Od tipova podataka C# sadrži logički tip (bool), znakovne tipove (char i string), cjelobrojne tipove bez predznaka (byte, uint, ulong i ushort), cjelobrojne tipove s predznakom (sbyte, short, int, long), brojčane tipove jednostruke/dvostruke preciznosti (float, double), i decimalni tip (decimal). C# sadrži 3 vrste operatora : aritmetičke (unarni : '++', '—', '+' i '-' ; binarni : '+', '-', '*', '/', '%'), logičke ('&', '|', '!', '^', '&&', '||'), bitovne operatore ('~', '<<' ,'>>'), operatore jednakosti

('==' i '!=') i operatore usporedbe ('<', '>', '<=', '>='). C# jezik je jednostavan i pogodan za programiranje većih aplikacija u MVC ili nekom drugom programerskom obrascu.

3.3.4 ASP.NET Web API

ASP.NET Web API je predložak unutar okruženja ASP.NET koji služi za izgradnju Web aplikacijskog programskog sučelja (API – *Application programming interface*). Aplikacijsko programsko sučelje je servis baziran na HTTP zahtjevima, a može se iskoristiti u raznim oblicima programske podrške kao što su : web aplikacije (aplikacije koje se obavljaju u preglednicima), mobilne aplikacije, desktop aplikacije (računalne aplikacije), te za internet objekata (IoT – *eng. Internet of Things*). Web API je servis koji se temelji na REST arhitekturi. REST (*eng. Representational State Transfer*) je arhitekturalni obrazac za kreiranje aplikacijskih servisa, a sastoji se od određenih ograničenja (*eng. Constraints*) koja se moraju ispoštovati pri njegovom projektiranju :

1) **Klijent-Server ograničenje** – razdvaja klijent od poslužitelja, pri čemu svaki od njih obavlja svoj dio posla. Poslužiteljski dio se bavi obradom zahtjeva i pohranom podataka, a klijentski dio služi za postavljanje zahtjeva, te mora biti napravljen kao dobro sučelje prema korisnicima.
2) **Neovisnost između svakog zahtjeva** (*eng. Stateless*) – svaki novi zahtjev koji klijentski dio postavlja (šalje) poslužiteljskom dijelu mora biti neovisan o prethodnim zahtjevima.
3) **Priručna memorija** (*eng. Cache*) – podaci unutar odgovora poslužitelja na zahtjev klijenta mogu se spremati u priručnu memoriju. Spremljene podatke klijent može ponovno koristiti za neki idući zahtjev.
4) **Jedinstveno sučelje** (*eng. Uniform Interface*) – poboljšava vidljivost interakcije poslužitelja i klijenta, te razdvaja arhitekturu u dva dijela (klijent i server).
5) **Slojevitost sustava** (*eng. Layered system*) – arhitektura je sastavljena od više hijerarhijskih slojeva, čime je omogućena hijerarhijska komunikacija (svaka komponenta može vršiti interakciju isključivo s jednim slojem). Klijent nije u mogućnosti spoznati komunicira li s krajnjim poslužiteljem ili s nekim od njegovih posrednika.
6) **Kod na zahtjev** (*eng. Code on Demand*) – poslužitelji mogu privremeno proširiti ili prilagoditi funkcionalnosti klijentske strane predajom ili izvršavanjem koda (npr. Java appleti ili Javascript kod).

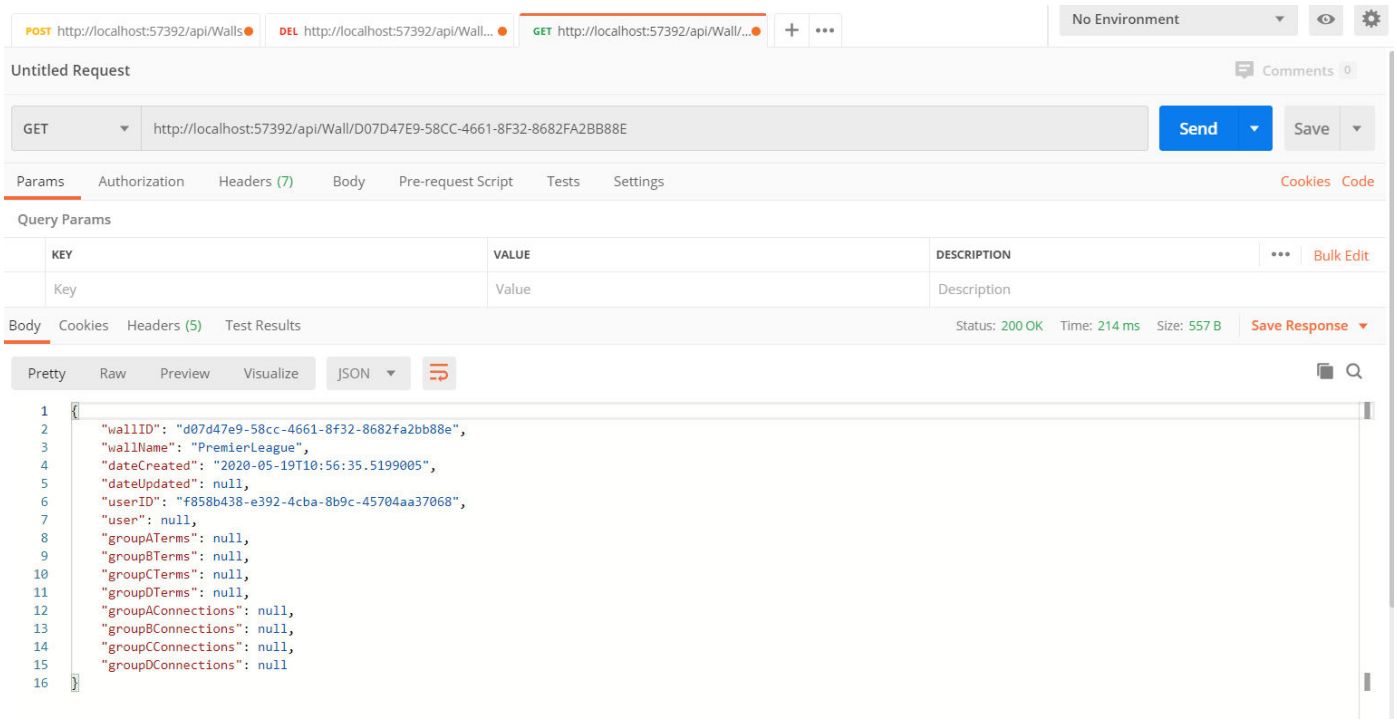
Komunikacija Web API-a između poslužitelja i klijentske strane radi na principu zahtjev/odgovor. Klijentska strana šalje ispravno formatiran HTTP zahtjev, te od poslužitelja dobiva odgovor u nekom od formata podataka (najčešće JSON ili XML). ASP.NET Web API radi po MVC obrascu za razvoj programske podrške, a HTTP zahtjevi koje klijentska strana šalje poslužitelju projektiraju se u upravljačima (kontrolerima). Kod razvoja i testiranja aplikacije pokretanje napravljenog Web API-a vrši se u IIS-u (*eng. Internet Information Services*), fleksibilnom web serveru koji prihvaća zahtjeve s lokalnog računala. Pokretanjem API-a u IIS-u, korisnik dobiva lokalnu adresu API-a na koju može slati HTTP zahtjeve iz neke druge aplikacije (klijentska aplikacija).

```
[HttpGet]
[Authorize]
//GET : /api/UserProfile
0 references
public async Task<Object> GetUserProfile()
{
    string userId = User.Claims.First(c => c.Type == "UserID").Value;
    var user = await _userManager.FindByIdAsync(userId);
    return new
    {
        user.FullName,
        user.Email,
        user.UserName,
        user.Id
    };
}
```

Slika 3.6. *Prikaz definiranog zahtjeva za dohvaćanje korisnika unutar kontrolera ASP.NET Web API-a.*

3.4 Postman

Postman je platforma za testiranje aplikacijskih programskih sučelja (API-a). Služi kao alat za ispitivanje HTTP zahtjeva koji su formirani na poslužiteljskoj strani, te pomaže u projektiranju zahtjeva na klijentskoj strani. Njegov razvoj počeo je Abhinav Asthana s ciljem kreiranja alata koji bi pojednostavio proces testiranja API-a. Zajedno sa svojim kolegama odlučio je napraviti okruženje koje će svima omogućiti jednostavnije testiranje API-a, a time i lakši razvoj složenih aplikacija. Njime je moguće testirati osnovne zahtjeve (GET, PUT POST, DELETE i HEAD), te ostale HTTP zahtjeve (OPTIONS, PATCH, LINK, UNLINK itd.) i dobiti odgovor u skladu s parametrima koje smo naveli.



Slika 3.7. *Prikaz testiranja GET zahtjeva u Postman platformi, te odgovora na zahtjev. Zahtjev koji testiramo mora prethodno biti definiran u kontroleru poslužiteljske strane.*

3.5 Angular 2+

Angular 2+ je besplatan okvir (*eng. Framework*) za razvoj web aplikacija baziran na TypeScript programskom jeziku. Okvir je namijenjen razvoju korisničke strane aplikacije (*eng. Frontend*), a njegovu prvu verziju AngularJS razvio je Google 2010. godine. AngularJS bio je baziran na JavaScript programskom jeziku i namijenjen za razvoj web aplikacija. Od druge verzije (Angular 2) razvijene 2016. godine okvir se značajno mijenja, MVC arhitekturu AngularJS-a zamjenjuju komponente (*eng. Components*) i smjernice (*eng. Directives*). Od verzije 2, Angular uz izgradnju web aplikacija omogućuje i izgradnju mobilnih aplikacija. Komponente Angular aplikacije predstavljaju pogled i logiku stranice, dok smjernice povezuju elemente objektnog modela dokumenta (DOM – *Document Object Model*) s podacima aplikacije. Osnovne značajke Angular 2+ (Angular 2 i novijih verzija) okvira su :

1) **Pristup gotovim predlošcima aplikacije** (*eng. Templates*), predlošci se sastoje od HTML elemenata, jezika za njihovo stiliziranje (CSS ili Sass) i TypeScript datoteke.

2) **Angular komponente** (*eng. Angular Components*) osnovna su gradivna jedinica svake aplikacije, web stranica može se razvijati tako da se svaki dio web stranice podijeli u više komponenti, gdje svaka sadrži HTML elemente koji se upravljaju Angular servisima (TypeScript datoteke)
3) Angular omogućuje **Router servis**, u kojem se mogu definirati reference na sve komponente koje se koriste na stranici. Također, omogućuje navigaciju između stanja aplikacije i njezine hijerarhije (komponenti)
4) **Angular moduli** (*eng. Angular Modules*) omogućuju učitavanje više funkcionalnih modula u aplikaciju (npr. Bootstrap modul za stiliziranje HTML elemenata ili HttpClient modul za komunikaciju s API-jima). Unutar glavnog modula (app.module) korisnici imaju pregled svih komponenti koje čine aplikaciju.
5) Postoje tri tipa povezivanja HTML elemenata s podacima aplikacije unutar TypeScript datoteka :
    a) **Povezivanje po događajima** (*eng. Event Binding*) – omogućuje odzive na korisnički unos tako što mijenja podatke u aplikaciji (npr. `(click)="myFunction($event)"` poziv definirane funkcije myFunction, povezivanje funkcije s DOM elementom).
    b) **Povezivanje po svojstvima** (*eng. Property Binding*) – postavljanje vrijednosti HTML elemenata na vrijednosti varijabli u nekoj komponenti (npr. `[value]="hello"` , gdje je hello varijabla definirana unutar komponente).
    c) **Obostrano povezivanje podataka** (*eng. Two-way data binding*) – omogućuje korištenje događaja s povezivanjem po svojstvima (Primjer :

```
<input [(ngModel)]="username">
<p> Hello {{username}}!</p>
```

Posljednja verzija Angular 2+ okvira izdana je u Lipnju 2020. godine, a od verzije 2 do aktualne verzije okvir se nije značajnije mijenjao po arhitekturi i sintaksi. Okvir pruža ugrađene biblioteke za izradu animacija, HTTP zahtjeva, te razne module za sortiranje, unos i strukturiranje podataka (npr. MatTable i BrowserAnimations moduli). Angular 2+ je idealan okvir za razvoj složenijih web aplikacija koji omogućuje efikasnu raspodjelu zasebnih dijelova aplikacije u komponente. Uz Vue.js i React, Angular je najpopularniji okvir za razvoj korisničkog dijela aplikacije.

Slika 3.8. *Prikaz Angular komponente za registraciju korisnika. Od servisa koristi korisnički definiranu UserService klasu i gotovu ToasterService klasu.*

3.5.1 HTML

HTML (*eng. HyperText Markup Language*) je prezentacijski (opisni) jezik za izradu dokumenata koji se prikazuju u web preglednicima. Web preglednici koriste HTML jezik za prikaz podataka u tekstualnom, slikovnom ili nekom drugom obliku. Osnovne komponente HTML jezika su HTML oznake (*eng. HTML tags*) koje se koriste za definiranje strukture sadržaja web stranice, odnosno određuju način na koji će sadržaj stranice biti prikazan. Osim oznaka koje definiranju prikaz strukture sadržaja, HTML se sastoji i od poveznica (*eng. links*) koje omogućuju povezivanje sadržaja jedne stranice sa sadržajem drugih stranica. HTML je razvio Tim Berners-Lee 1991. godine za prijenos informacija, a prva verzija dostupna javnosti bila je HTML 2.0.

Aktualna verzija HTML-a je HTML 5.2 izdana 2014. godine koja sadrži značajke za upravljanje raznim multimedijalnim sadržajem (video, audio, slikarsko platno itd.). HTML datoteke obične su tekstualne datoteke koje započinju linijom <!DOCTYPE html> i sadrže .html ekstenziju. Ovakve datoteke podržavaju svi internetski preglednici. Osnovni dijelovi HTML dokumenta su :

1) **Zaglavlje** – Između oznaka <head> i </head>. U zaglavlju se definiraju „podaci o podacima“, odnosno naziv dokumenta, poveznica na CSS dokument, skup znakova koji se koristi, te skriptu koja se koristi za omogućavanje događaja na stranici (JavaSript, Typescript ili druge).
2) **Tijelo** – Između oznaka <body> i </body>. Tijelo HTML dokumenta sadrži naslove, paragrafe, slike, poveznice (*eng. Hyperlinks*), liste, tablice itd. Može postojati samo jedno tijelo unutar svakog HTML dokumenta.

3.5.2 CSS

CSS (*eng. Cascading Style Sheets*) je jezik za stiliziranje sadržaja unutar HTML dokumenta. U početku su se HTML elementi stilizirali unutar HTML oznaka, no rezultat toga bio je nepregledan kod koji je bilo potrebno podijeliti na kod koji opisuje sadržaj stranice (HTML dokument) i kod koji određuje stilove sadržaja iz HTML dokumenta (CSS dokument). CSS je prvi put definiran 1996. godine, no razlike u web preglednicima u to vrijeme nisu omogućavale dosljedan i jednak prikaz sadržaja u svim preglednicima. Prošlo je dosta vremena do standardizacije CSS jezika, što se prvi put ostvarilo 2005. godine objavom standarda CSS 2.0, na kojoj su se temeljile daljnje verzije CSS-a. CSS datoteke omogućuju određivanje stila sadržaja unutar HTML oznaka mijenjajući im boju, položaj, font i veličinu slova, te mnoge druge značajke. CSS su tekstualne datoteke koje sadrže .css ekstenziju, a za omogućavanje stilizacije HTML dokumenta potrebno je dodati referencu na njih dokument unutar HTML <head> oznake. Postoji 3 pristupa kod stilizacije HTML dokumenta pomoću CSS opisnog jezika :

1) **Eksterni CSS** – Stiliziranje HTML dokumenta unutar zasebne datoteke koja sadrži .css ekstenziju, nužno je referencirati .css datoteku unutar HTML dokumenta. Moguće je posebno stiliziranje HTML elemenata s određenim identifikatorima i klasama, odnosno podjela elemenata u klase koje će se jednako stilizirati.
2) **Interni CSS** - stilizacija HTML elemenata između oznaka <style> i </style> u CSS jeziku.

3) **CSS unutar linije** (*eng. Inline CSS*) – CSS stiliziranje svakog pojedinog HTML elementa unutar njegovih oznaka.

Ako se za stiliziranje HTML dokumenata koriste sva tri pristupa istovremeno, najveći prioritet ima CSS unutar linije, pa tek onda interni i eksterni CSS.

3.5.3 TypeScript

TypeScript (skraćeno od Typed JavaScript) besplatan je programski jezik kojeg je razvio Microsoft. TypeScript je nadskup JavaScripta koji značajkama JavaScripta dodaje statičke tipove podataka koji omogućuju lakše razumijevanje i čitljivost koda. TypeScript je objektno-orijentiran jezik, dok je JavaScript skriptni jezik. Kod pisanja funkcija TypeScript omogućuje definiranje neobaveznih parametara što nije moguće u JavaScriptu. Osim razvoja klijentske strane aplikacije, TypeScript omogućuje i razvoj poslužiteljske strane aplikacije. Prva verzija TypeScript jezika razvijena je 2012. Godine, nakon čega su izdane još dvije verzije i brojne podverzije (trenutna verzija 3.9.5). Sintaksa TypeScripta slična je drugim objektno-orijentiranim jezicima (C#, C++, Java itd.), a osnovni tipova podataka koje sadrži su : logički tip (boolean), brojčani tip (number), znakovni tip (string), struktura nabrojanih elemenata (enum), proizvoljni tip (any), objekt i ostali tipovi (null, undefined itd.). TypeScript omogućuje kreiranje vlastitog tipa podatka, to jest sučelja objekta (*eng. Interface*). Skripte napisane u TypeScript jeziku se pri izvođenju uvijek prevode u JavaScript. TypeScript datoteke sastavni su dio Angular okruženja i one omogućuju izvođenje događaja korisničkog sučelja i povezivanje komponenti okruženja.

```
export class GroupConnection {
    connectionId:string;
    connectionName:string;

    constructor(id,name)
    {
        this.connectionId=id;
        this.connectionName=name;
    }
}
```

Slika 3.9. *Prikaz definiranja klase u TypeScriptu.*

3.5.4 NGX Bootstrap

NGX Bootstrap je besplatan alat koji omogućuje lakšu izradu složenijih web aplikacija. Prije opisivanja NGX Bootstrapa, potrebno je opisati Bootstrap. Bootstrap je besplatan i javno

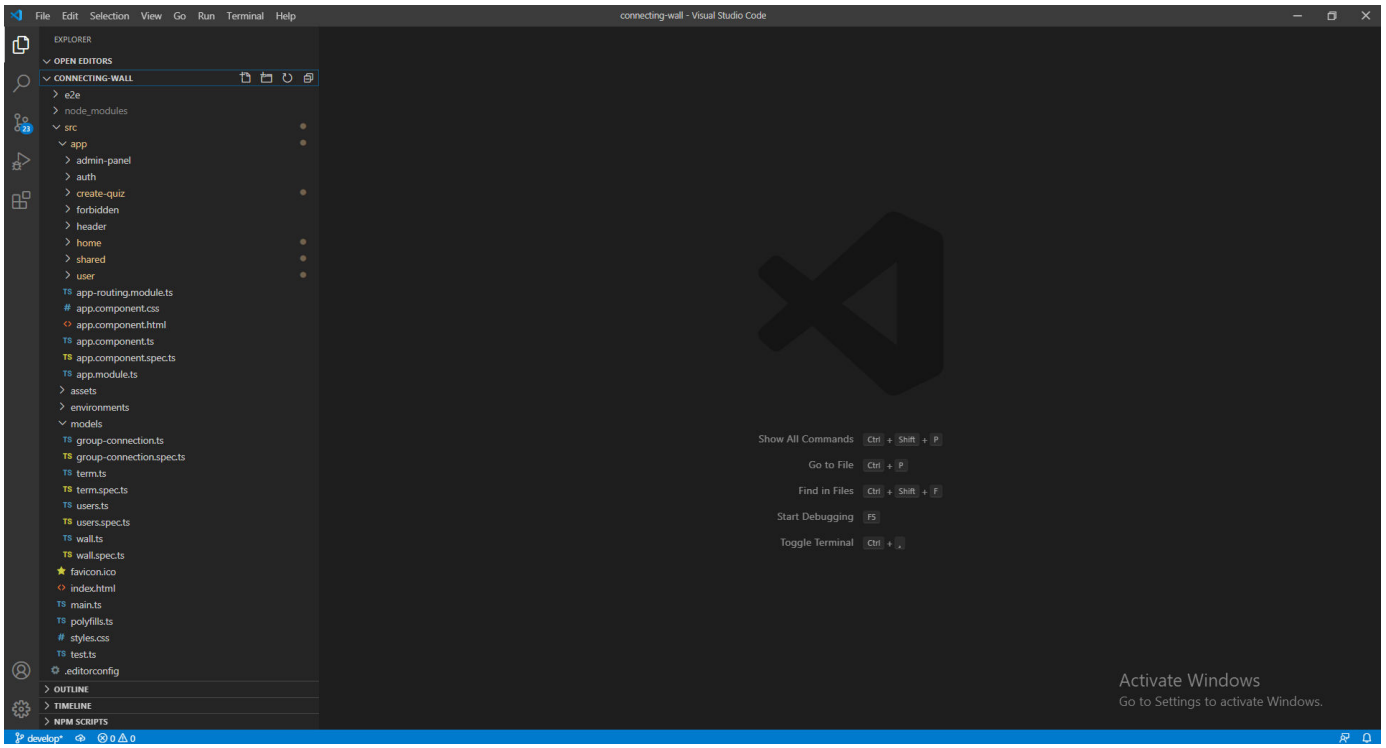
dostupan CSS okvir (*eng. Framework*) za razvoj korisničkog dijela web aplikacija (*eng. frontend*). Sastoji se od gotovih predložaka HTML-a, CSS-a i JavaScripta kojima se programeri mogu služiti u stiliziranju svoje aplikacije bez pisanja vlastitog CSS koda. Korištenjem Bootstrap okvira razvijaju se aplikacije prilagođene mobilnim uređajima (*eng. mobile-first*) i responzivne web aplikacije (aplikacije čiji je izgled prilagođen različitim rezolucijama). Bootstrap okruženje omogućuje programerima korištenje gotovih klasa za koje postoji gotov CSS kod, dovoljno je navesti da HTML element pripada gotovoj klasi i element će biti stiliziran na svakoj rezoluciji. Unutar Bootstrapa dostupne su razne gotove klase (predlošci) kojima se određuje stil fontova teksta, korištenih formi, gumbova, navigacije i ostalih komponenti sučelja stranice. NGX Bootstrap je verzija Bootstrapa prilagođena razvoju u Angular 2+ okruženju.

```
<app-header></app-header>
<div class="card m-5" style="width:20rem" *ngIf="userDetails">
<ul class="list-group">
    <li class="list-group-item"><strong>Username : </strong>{{userDetails.userName}}</li>
    <li class="list-group-item"><strong>Full name : </strong>{{userDetails.fullName}}</li>
    <li class="list-group-item"><strong>E-mail : </strong>{{userDetails.email}}</li>
</ul>
</div>
```

Slika 3.10. *Prikaz korištenja gotovih Bootstrap klasa (card i list group) i podklasa (list-group-item) za stiliziranje HTML elemenata.*

3.6 Visual Studio Code

Visual Studio Code je besplatan alat za uređivanje koda kojeg je razvila tvrtka Microsoft. Alat omogućuje podršku za uklanjanje neispravnosti u kodu (*eng. Debugging*), označavanje sintakse, podršku u razvoju za velik broj programskih jezika (Python, C/C++, C#, Java, Ruby, HTML i mnogi drugi), te inteligentno nadopunjavanje koda. Dostupna je instalacija brojnih dodatnih alata (*eng. extensions*) koji pomažu u razvoju aplikacija, kao i naredbeni redak za lakšu manipulaciju brojnim datotekama u razvoju složenijih aplikacija. Prva verzija Visual Studio Code-a objavljena je 2015. godine uz pod blagim MIT odobrenjem (veća količina stavki programske podrške je besplatna). Prema podacima sa Stack Overflow zajednice programera, Visual Studio Code najpopularniji je alat u razvoju programske podrške.



Slika 3.11. *Prikaz početnog sučelja Visual Studio Code-a. U pregledniku alata je prikazana struktura projekta na kojem se radi.*

4. IZRADA APLIKACIJE

4.1 Izrada relacijskog modela podataka

Za problem „Zid za spajanje“ potrebno je napraviti normalizirani model podataka koji se sastoji od entiteta i njihovih međusobnih veza.

4.1.1 Zadani problem

Aplikacija ima više korisnika gdje svaki korisnik može kreirati jedan ili više zidova za spajanje, odnosno svaki zid za spajanje pripada jednom korisniku. Korisnik se sastoji od identifikatora, korisničkog imena, e-maila, potvrđenog e-maila, broja telefona, punog imena i mnogih drugih atributa iz gotove klase IdentityUser koja je dio ASP.NET okruženja. Zid za spajanje sadrži identifikator, ime, datum kreiranja, datum izmjene, te 4 liste koje predstavljaju pojmove grupa A, B, C i D. Također, sadrži i 4 liste koje predstavljaju izraze po kojima su vezani pojmovi grupa. Pojam se sastoji od identifikatora i imena. Grupni izraz se također sastoji od identifikatora i imena. Zid se sastoji od 16 pojmova i neodređenog broja izraza po kojima su vezani pojmovi svake grupe.

4.1.2 Utvrđivanje i analiza normaliziranog modela podataka (ER modela) „Zida za spajanje“

ENTITETI : **ApplicationUser** (korisnik), **Wall** (zid), **Term** (pojam zida), **GroupConnections** (izraz po kojemu su grupirani pojmovi)

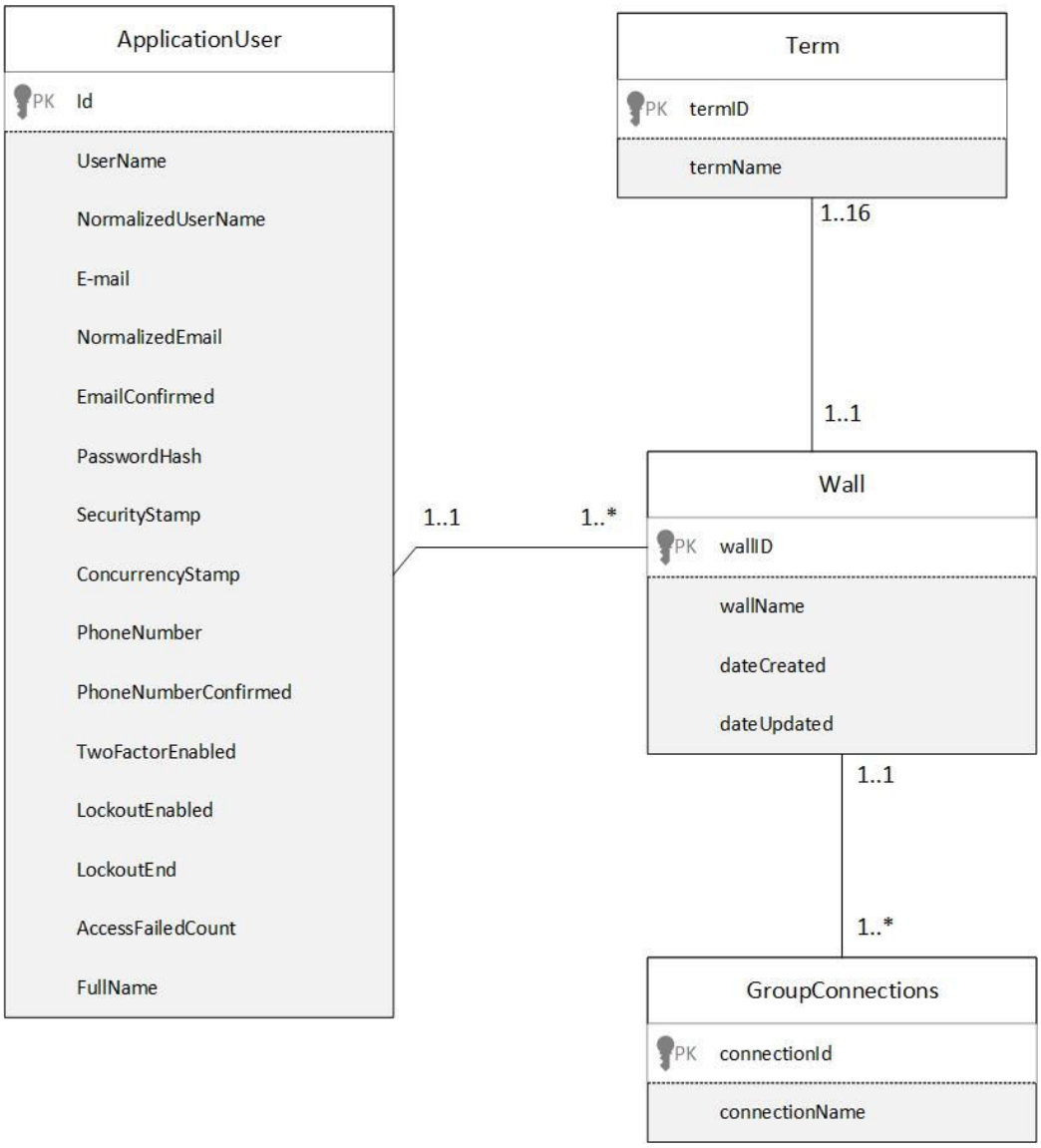
ATRIBUTI ENTITETA (podcrtani atribut je primarni ključ – atribut koji je jedinstven svakoj instanci entiteta) :

ApplicationUser (Id, UserName, NormalisedUserName, Email, NormalizedEmail, EmailConfirmed, PasswordHash, SecurityStamp, ConcurrencyStamp, PhoneNumber, PhoneNumberConfirmed, TwoFactorEnabled, LockoutEnd, LockoutEnabled, AccessFailedCount, FullName)

Wall (wallID, wallName, dateCreated, dateUpdated)

Term (termID, termName)

GroupConnections (connectionId, connectionName)



Slika 4.1. *Entity-Relationship model aplikacije „Zid za spajanje“ izrađen u programu Microsoft Visio.*

4.1.3 Pretvaranje ER modela u relacijski model podataka

Kako bi se izradio model aplikacije po stvorenom modelu, potrebno je pretvoriti postojeći Entity-Relationship model podataka u relacijski model podataka. To podrazumijeva primjenu teorije modeliranja podataka. Prema toj teoriji, entiteti koji su spojeni vezom jedan-prema-više će u relacijskom modelu biti prikazani tako da će se primarni ključevi entiteta na strani jedan dodavati kao strani ključevi u entitete na strani više. Ostali tipovi veza (više-prema-više i jedan-

prema-jedan) nisu prisutne u Entity-Relationship modelu ove aplikacije. Slijedi prikaz Entiteta i atributa entiteta nakon pretvaranja Entity-Relationship modela u relacijski model podataka.

ENTITETI : **ApplicationUser** (korisnik), **Wall** (zid), **Term** (pojam zida), **GroupConnections** (izraz koji spaja pojmove u grupi)

ATRIBUTI ENTITETA (podcrtani atribut je primarni ključ – atribut koji je jedinstven svakoj instanci entiteta, zadnji navedeni atribut predstavlja strani ključ koji je referenca na jedinstvenu instancu drugog entiteta) :

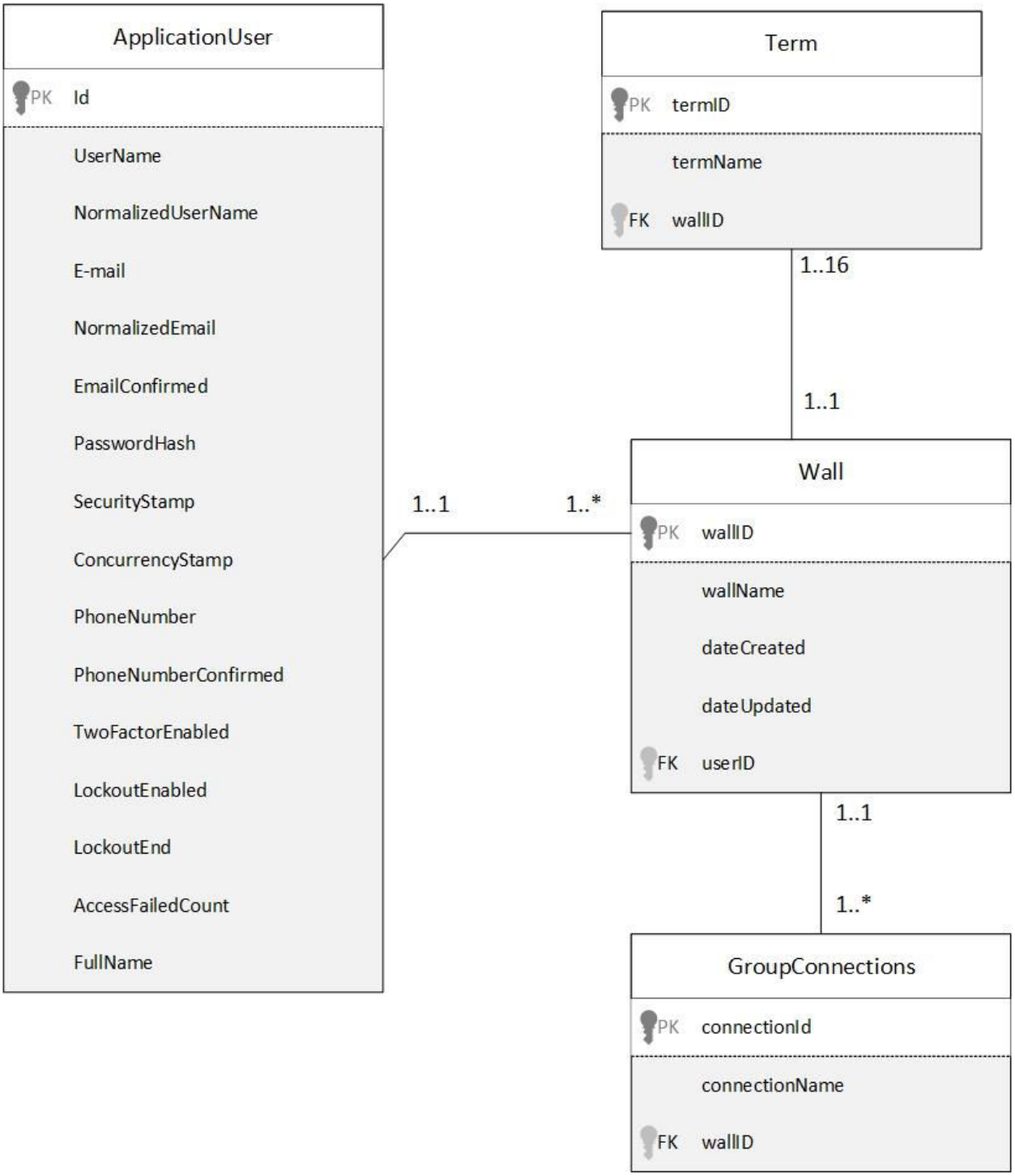
ApplicationUser (Id, UserName, NormalisedUserName, Email, NormalizedEmail, EmailConfirmed, PasswordHash, SecurityStamp, ConcurrencyStamp, PhoneNumber, PhoneNumberConfirmed, TwoFactorEnabled, LockoutEnd, LockoutEnabled, AccessFailedCount, FullName)

Wall (wallID, wallName, dateCreated, dateUpdated, userID)

Term (termID, termName, wallID)

GroupConnections (connectionId, connectionName, wallID)

Napomena : Pošto će se model podataka raditi Code-First pristupom unutar ASP.NET okruženja, format podataka će biti nešto drukčiji jer okruženje sadrži određena pravila kod kreiranja modela i raspoređivanja atributa entiteta. Detaljan opis pravila i modela podataka unutar ASP.NET okruženja nalazi se u poglavlju 4.2.



Slika 4.2. *Prikaz relacijskog modela aplikacije „Zid za spajanje“ koji je izrađen u programu Microsoft Visio.*

4.2 Izrada poslužiteljske aplikacije u .NET okruženju

U ovom poglavlju će se opisati spajanje aplikacije (Entity Framework Web API) s bazom podataka, izrada modela u Entity Framework okviru i izgradnja tablica u bazi na temelju tog modela, te definiranje upravljačkih funkcija (formatiranje HTTP zahtjeva koji će se slati s korisničkog sučelja). Također će se opisati način autentikacije korisnika i brisanje n-torki iz baze podataka.

4.2.1 Spajanje aplikacije s bazom podataka

Nakon određivanja relacijskog modela podataka, stvara se projekt u Visual Studiu koji će služiti kao web poslužitelj (Web API). Aplikacija je bazirana na MVC programerskom obrascu, te je prema njemu izvršena podjela koda. Osim modela i kontrolera koji su dio Visual Studio Web aplikacije, potrebno je postaviti bazu podataka nad kojom će vršiti operacije stvaranja, čitanja, ažuriranja i brisanja podataka (CRUD operacije). Kao što je spomenuto u poglavlju 3., korištena je SQL relacijska baza podataka, a za njezin pregled i upravljanje koristi se aplikacija Microsoft SQL Server Management Studio. Prvi korak kod spajanja Web API-a s lokalnom bazom podataka je određivanje niza znakova za spajanje (*eng. Connection String*). Konfiguracija tog niza znakova obavlja se u dokumentu appsettings.json gdje se navodi ime lokalnog SQL poslužitelja (Server), ime baze podataka koja se koristi (Database), potvrda da je spajanje baze i aplikacije dozvoljeno (*eng. Trusted_Connection*), te potvrda da je dozvoljeno unositi više objekata odjednom (*eng. MultipleActiveResultSets*). Kod pokretanja aplikacije, nužno je izvršiti naredbu koja određuje koja se baza podataka koristi, te niz znakova za spajanje s bazom (oboje unutar dokumenta Startup.cs).

Slika 4.3. *Konfiguriranje niza znakova za spajanje s bazom podataka.*

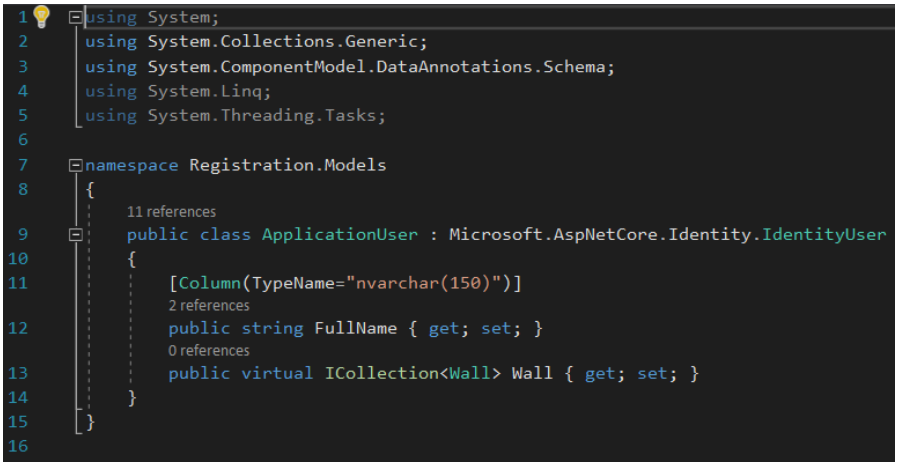
```
public void ConfigureServices(IServiceCollection services)
{
    //for CORS Policy

    //injecting AppSettings

    services.Configure<ApplicationSettings>(Configuration.GetSection("ApplicationSettings"));

    CRUDOperations

    services.AddControllers();
    services.AddControllers().AddNewtonsoftJson(options =>
    options.SerializerSettings.ReferenceLoopHandling = Newtonsoft.Json.ReferenceLoopHandling.Ignore);
    services.AddDbContext<Models.AuthenticationContext>(options =>
    options.UseSqlServer(Configuration.GetConnectionString("IdentityConnection"))
    );
```

Slika 4.4. *Kod unutar Startup.cs dokumenta kojim se određuje veza i svojstva veze s bazom podataka.*

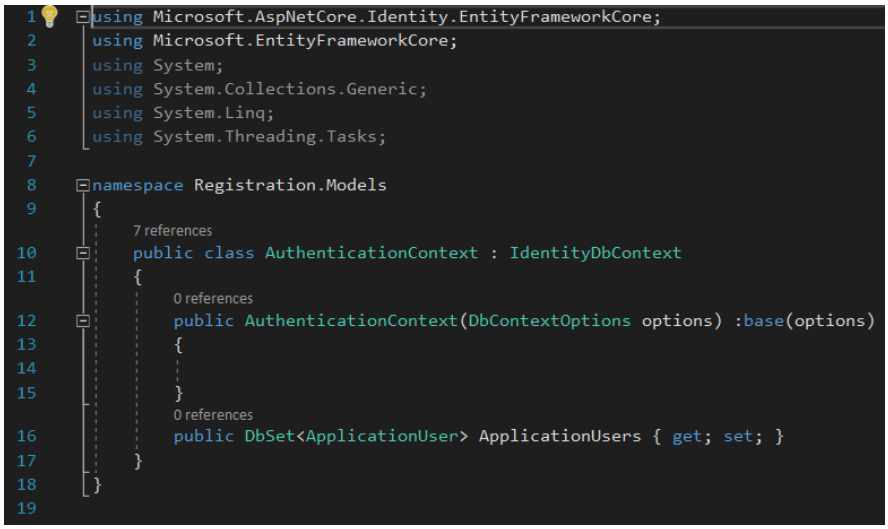
4.2.2 Model podataka u .NET okruženju

Kod izrade tablica baze podataka koristi se Code-First pristup Entity Framework okvira. Takav pristup svodi se na stvaranje entiteta-klasa na temelju relacijskog modela podataka aplikacije koja se stvara. Relacijski model aplikacije „Zid za spajanje“ dan je na slici 4.1.3. Pretvaranje entiteta-klasa Entity Framework okvira u ekvivalentne tablice unutar SQL baze podataka vrši se preko Entity Framework Migracija (*eng. Migrations*). Migracije omogućavaju programerima obostrano upravljanje aplikacijskim modelom i bazom podataka, te održavanje njihovog međuodnosa. Programeri jednostavno mogu promijeniti modele podataka svojih aplikacija preko izvršavanja migracija, bez ulaganja dodatnog napora za posebno mijenjanje modela podataka u bazi. Kod izrade aplikacije koristile su se dvije odvojene migracije – prva migracija je pretvarala klasu AplicationUser u tablice baze podataka (tablica korisnik i posebne tablice koje bilježe

korisničke zahtjeve, uloge, prijave i tokene), a druga migracija je pretvarala entitete-klase Wall, GroupConnections i Terms u ekvivalentne SQL tablice. Za omogućavanje izvođenja migracija, potrebno je stvoriti posebne kontekst klase (*eng. Database Context Class*) koje omogućuju pohranu podataka u bazu, te upite nad bazom podataka preko aplikacijskog modela u Entity Framework okruženju (upiti u kontrolerima). Definiranje migracija i stvaranje tablica u bazi podataka iz klasa u Entity Framework okviru obavlja se unutar Package Manager konzole preko naredbi Add-Migration i Update-Database, pri čemu je obavezno navesti kontekst klasu koja se koristi ako postoji više njih.



Slika 4.5. *Klasa ApplicationUser koja nasljeđuje IdentityUser gotovu klasu.*



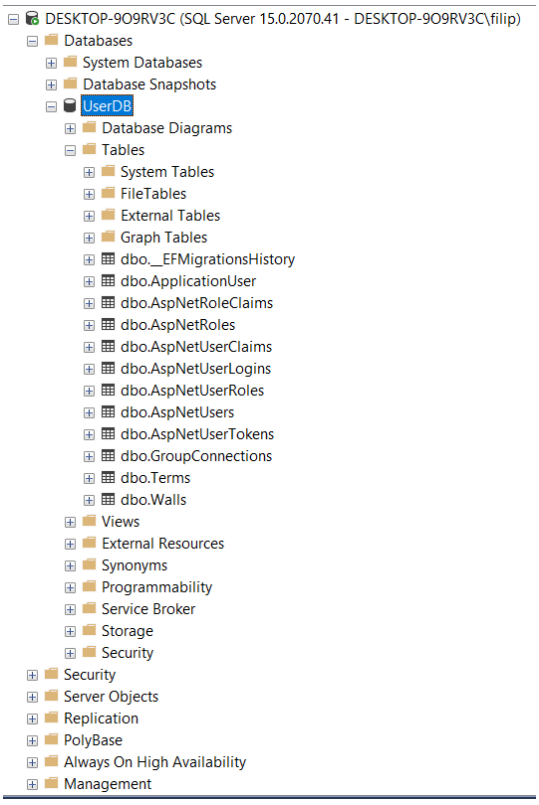
Slika 4.6. *Kontekst klasa po kojoj se stvara tablica ApplicationUser.*

```
1.  using Microsoft.EntityFrameworkCore;
2.  using System;
3.  using System.Collections.Generic;
4.  using System.Linq;
5.  using System.Security.Cryptography.X509Certificates;
6.  using System.Text.RegularExpressions;
7.  using System.Threading.Tasks;
8.
9.  namespace Registration.Models
10. {
11.     public class APIDBContext : DbContext
12.     {
13.         public DbSet<ApplicationUser> AspNetUsers { get; set; }
14.         public DbSet<Wall> Walls { get; set; }
15.         public DbSet<GroupConnections> GroupConnections { get; set; }
16.         public DbSet<Term> Terms { get; set; }
17.         public APIDBContext(DbContextOptions<APIDBContext> options) : base(options)
18.         {
19.
20.         }
21.     protected override void OnModelCreating(ModelBuilder mb)
22.      {
23.
24.        base.OnModelCreating(mb);
25.        mb.Entity<Wall>().Property(x => x.wallID).HasDefaultValueSql("NEWID()");
26.        mb.Entity<Term>().Property(x => x.termID).HasDefaultValueSql("NEWID()");
27.        mb.Entity<GroupConnections>().Property(x => x.connectionId).HasDefaultValueSql("
    NEWID()");
28.
29.             mb.Entity<Wall>()
30.                 .HasKey(s => s.wallID);
31.
32.             mb.Entity<Term>()
33.                 .HasKey(s => s.termID);
34.
35.             mb.Entity<GroupConnections>()
36.                 .HasKey(s => s.connectionId);
37.
38.      }
39.     }
40. }
```

Slika 4.7. *Kontekst klasa po kojoj se stvaraju tablice Wall, Term i GroupConnections. Naredba HasDefaultValueSql određuje novi identifikator svake kreirane instance.*

```
1.  public class Wall
2.      {
3.          [Key]
4.          public Guid wallID { get; set; }
5.          [Required]
6.          public string wallName { get; set; }
7.          public DateTime dateCreated { get; set; } = DateTime.Now;
8.          public DateTime? dateUpdated { get; set; }
9.          [ForeignKey("userID")]
10.         public string userID { get; set; }
11.         public virtual ApplicationUser User { get; set; }
12.
13.         public List<Term> groupATerms { get; set; }
14.         public List<Term> groupBTerms { get; set; }
15.         public List<Term> groupCTerms { get; set; }
16.         public List<Term> groupDTerms { get; set; }
17.         public List<GroupConnections> groupAConnections { get; set; }
18.         public List<GroupConnections> groupBConnections { get; set; }
19.         public List<GroupConnections> groupCConnections { get; set; }
20.         public List<GroupConnections> groupDConnections { get; set; }
21.
22.     }
```

Slika 4.8. *Prikaz definirane klase Wall (zid) iz koje se stvara tablica Zid u bazi podataka.*



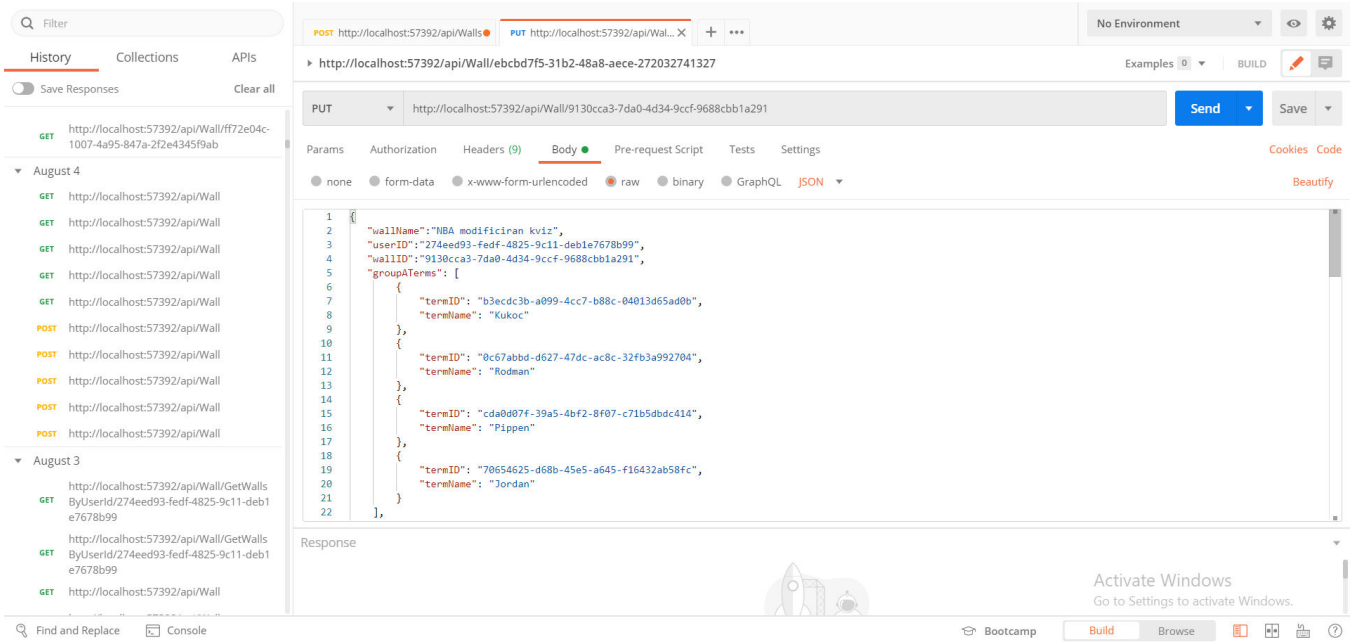
Slika 4.9. *Prikaz napravljenih tablica u bazi podataka nakon izvedenih migracija.*

4.2.3 Definiranje upravljača

Za prihvat i obradu zahtjeva s korisničke strane aplikacije nužno je napraviti upravljačke (kontroler) klase. Standardna MVC konvencija propisuje da svaki model ima svoju kontroler klasu. Kod izrade aplikacije koristila su se tri upravljača : ApplicationUserController, UserProfileController i WallController. ApplicationUserController služi za registraciju i prijavu korisnika, UserProfileController za brisanje i dohvaćanje korisničkog profila, a WallController za stvaranje, dohvaćanje, modifikaciju i brisanje zidova. HTTP zahtjevi koji će se slati s korisničkog sučelja zahtijevaju dohvaćanje svih zidova, dohvaćanje zida po identifikatoru zida, dohvaćanje zida po korisničkom identifikatoru, pohrana zida, izmjena postojećeg zida po identifikatoru, te brisanje zida po identifikatoru. Stvaranjem kontroler klase navodi se model za koji se upravljač definira, nakon čega se definiraju osnovne upravljačke funkcije (dohvaćanje, unos, izmjena i brisanje po identifikatoru). Također, kod definiranja upravljača nužno je instancirati objekt kontekst klase koja se koristi za operacije izmjene podataka. Svaka upravljačka funkcija mora imati određen put (*eng. path*) preko kojeg joj korisnička strana pristupa. Ako programer korisničke strane želi pristupiti podacima iz baze podataka (ili ih unijeti), on mora ispravno navesti put HTTP zahtjeva zajedno s podacima koje određena upravljačka funkcija zahtjeva. Na primjer, za definiranje upravljačke funkcije koja dohvaća sve zidove za spajanje od određenog korisnika, unutar jedinstvenog lokatora resursa (*eng. Uniform Resource Locator*) potrebno je osim adrese Web API-a navesti i identifikator korisnika. Adrese predefiniranih upravljačkih funkcija također su unaprijed određene, a ako programer stvara vlastitu upravljačku funkciju, obavezan je navesti HTTP adresu na koju se šalje zahtjev. Da bi se ispravno formatirao zahtjev, te da bi se provjerilo šalje li API željene podatke (i u poželjnom formatu) potrebno je testiranje zahtjeva u aplikaciji Postman. Napravljene upravljačke funkcije definiraju HTTP zahtjeve koji će se slati s korisničke strane aplikacije (*eng. Frontend*), a ti zahtjevi će biti opisani u poglavlju 4.3. Entity Framework Web API zbog sigurnosnih razloga ne dozvoljava izmjenu i pristup podacima sa svakog URL-a, zbog čega se unutar Startup dokumenta mora navesti adresa korisničkog servera s kojeg su zahtjevi dozvoljeni.

```
1.  // GET: api/Wall //predefinirani URL
2.          [HttpGet]
3.          public async Task<ActionResult<IEnumerable<Wall>>> GetWalls()
4.          {
5.              return await _context.Walls
6.                  .Include(p=>p.User)
7.                  .Include(p=>p.groupATerms)
8.                  .Include(p=>p.groupBTerms)
9.                  .Include(p => p.groupCTerms)
10.                 .Include(p => p.groupDTerms)
11.                 .Include(p => p.groupAConnections)
12.                 .Include(p=>p.groupBConnections)
13.                 .Include(p => p.groupCConnections)
14.                 .Include(p => p.groupDConnections)
15.                 .ToListAsync();
16.
17.         }
18.         [HttpGet("GetWallsByUserId/{userId}")]  //Navođenje posebnog URL-a
19.         public async Task<ActionResult<IEnumerable<Wall>>> GetWallsByUserId(string user
    Id)
20.         {
21.             return await _context.Walls
22.                 .Where(w => w.userID == userId)
23.                 .Include(p=>p.User)
24.                 .Include(p => p.groupATerms)
25.                 .Include(p => p.groupBTerms)
26.                 .Include(p => p.groupCTerms)
27.                 .Include(p => p.groupDTerms)
28.                 .Include(p => p.groupAConnections)
29.                 .Include(p => p.groupBConnections)
30.                 .Include(p => p.groupCConnections)
31.                 .Include(p => p.groupDConnections)
32.                 .ToListAsync();
33.         }
```

Slika 4.10. *Upravljačke funkcije za dohvaćanje svih zidova za spajanje i dohvaćanje svih zidova po korisničkom identifikatoru.*



Slika 4.11. *Prikaz testiranja PUT zahtjeva (zahtjeva za izmjenu ili stvaranje podataka) u programu Postman.*

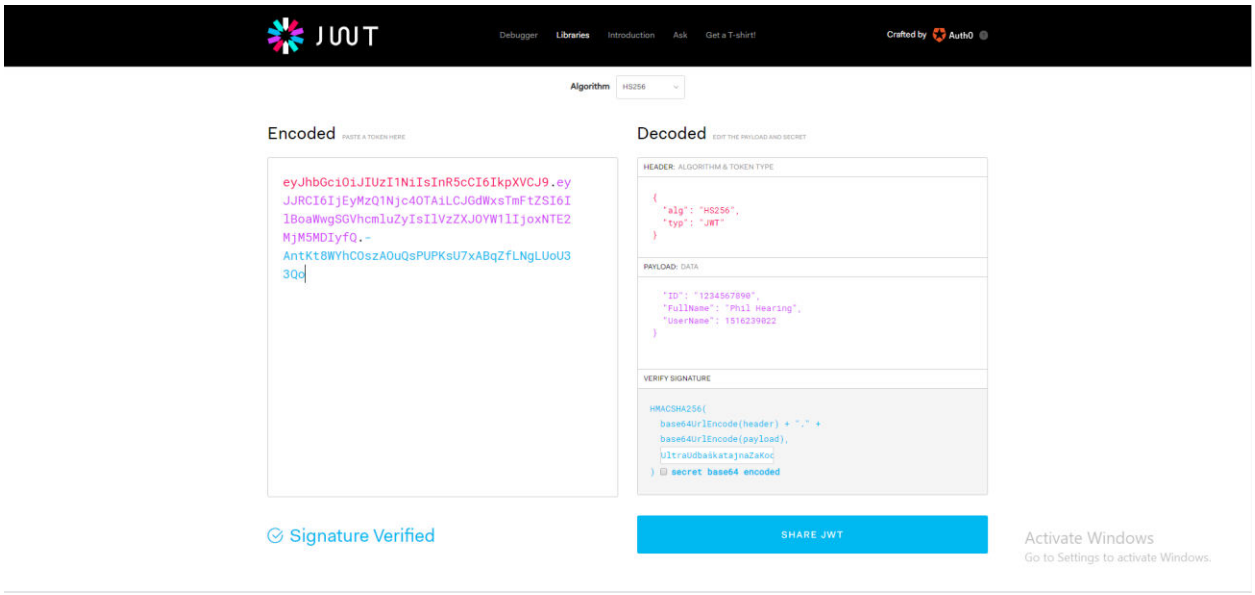
```
1.  // This method gets called by the runtime. Use this method to configure the HTTP reques
    t pipeline.
2.          public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
3.          {
4.              if (env.IsDevelopment())
5.              {
6.                  app.UseDeveloperExceptionPage();
7.              }
8.              app.UseCors(builder =>
9.     builder.WithOrigins(Configuration["ApplicationSettings:ClientURL"].ToString()).Allow
    AnyHeader().AllowAnyMethod() // Određivanje adrese s koje su zahtjevi dozvoljeni
10.             );
11.             app.UseAuthentication();
12.             app.UseRouting();
13.             app.UseAuthorization();
14.             app.UseEndpoints(endpoints =>
15.             {
16.                 endpoints.MapControllers();
17.             });
18.         }
19.     }
20. }
```

Slika 4.12. *Funkcija unutar Startup dokumenta koja omogućuje HTTP zahtjeve prema API-u s ClientURL adrese. ClientURL vrijednost dana je unutar ApplicationSettings datoteke.*

4.2.4 Korisnička autentikacija

Model aplikacije sadrži entitet korisnik, zbog čega aplikacija mora omogućiti registraciju i prijavu korisnika koji može kreirati, mijenjati i brisati vlastite zidove za spajanje i igrati tuđe kvizove. Kod registracije i prijave lozinku korisnika potrebno je pohraniti na siguran način (učiniti je nevidljivom administratoru baze podataka) korištenjem određene vrste enkripcije podataka. Za siguran prijenos osjetljivih korisničkih podataka koristit će se JSON Web Token, otvoreni standard koji omogućuje slanje podataka preko Jedinstvenog lokatora resursa, HTTP Post zahtjeva ili HTTP zaglavlja. Svaki Web Token sastoji se od zaglavlja (*eng. header*), tijela (*eng. payload*) i potpisa (*eng. signature*). Zaglavlje tokena sadrži informacije o algoritmu po kojem se oblikuje token i njegov tip (JWT), tijelo sadrži podatke o korisniku (osnovni podaci o korisniku i vrijeme isteka tokena), a potpis se sastoji od spoja tijela i zaglavlja s dodanom tajnom riječi za kodiranje. Kad se korisnik aplikacije prijavi, poslat će se zahtjev za korisničkim podacima Web API-u, a podaci će se vratiti i pohraniti u lokalnu memoriju preglednika kao Token (ako korisnik postoji). Za korištenje JWT Tokena potrebno je dodati autentikaciju unutar Startup dokumenta, te dodati Token kod HTTP Post zahtjeva za prijavu korisnika unutar ApplicationUserController upravljačke klase. Naposljetku, kod upravljača koji dohvaća korisničke podatke (UserProfileController) potrebno je navesti obaveznu autentikaciju u obliku

JWT Tokena. Nakon obavljenih operacija slijedi testiranje JWT Token autentikacije korisnika kroz Postman aplikaciju, a zatim i kroz korisničku aplikaciju.



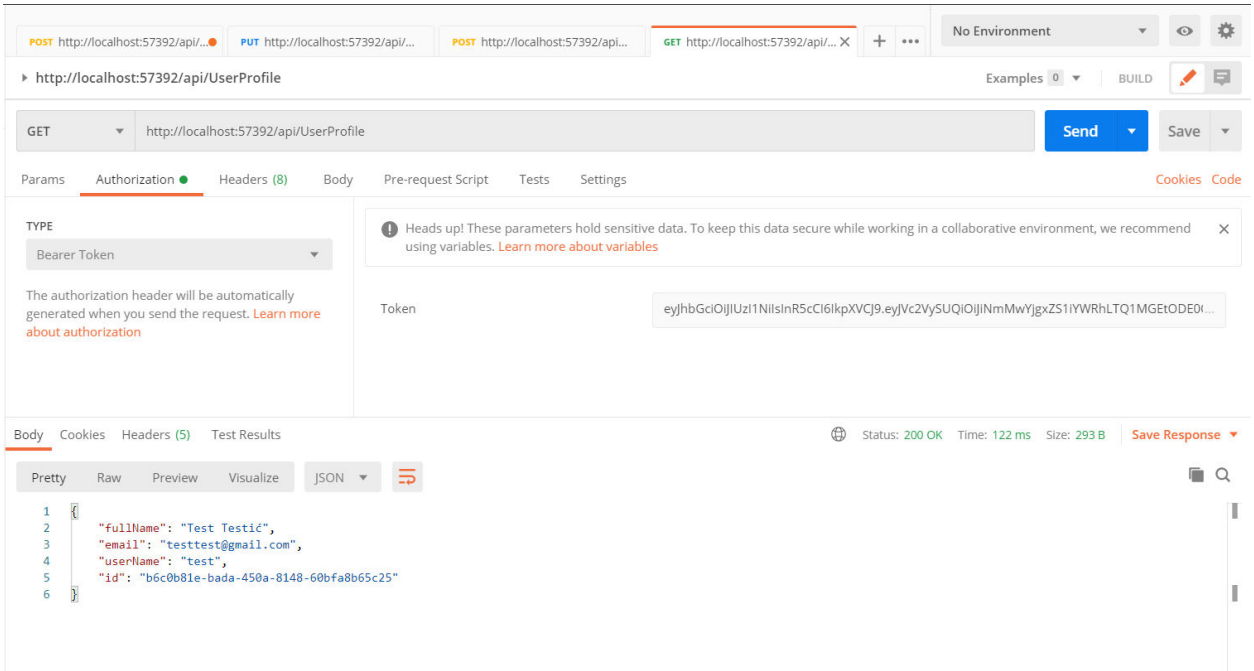
Slika 4.13. *Demonstracija kreiranja sadržaja Tokena iz zadanih korisničkih podataka [20].*

```
1.  //JWT authentication, secret part
2.  var key =Encoding.UTF8.GetBytes(Configuration["ApplicationSettings:JWT_Secret"].ToStrin
    g());// ključ autentikacije, pristupanje tajnom nizu znakova
3.              services.AddAuthentication(x=>
4.              {
5.                  x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
6.                  x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
7.                  x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
8.              }
9.              ).AddJwtBearer(x=>
10.             {
11.                 x.RequireHttpsMetadata = false;  //bez spremanja adrese
12.                 x.SaveToken = false; //bez spremanja tokena
13.                 x.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenV
    alidationParameters
14.                 {
15.                     ValidateIssuerSigningKey = true,  //validacija ključa
16.                     IssuerSigningKey = new SymmetricSecurityKey(key), //kreiranje JWT
    tokena
17.                     ValidateIssuer = false, //ne validira se server s kojeg je token
18.                     ValidateAudience = false, //ne validira se server koji prima token
19.                     ClockSkew = TimeSpan.Zero  //vrijeme stvaranja tokena
20.                 };
21.             }
22.             );
23.         }
```

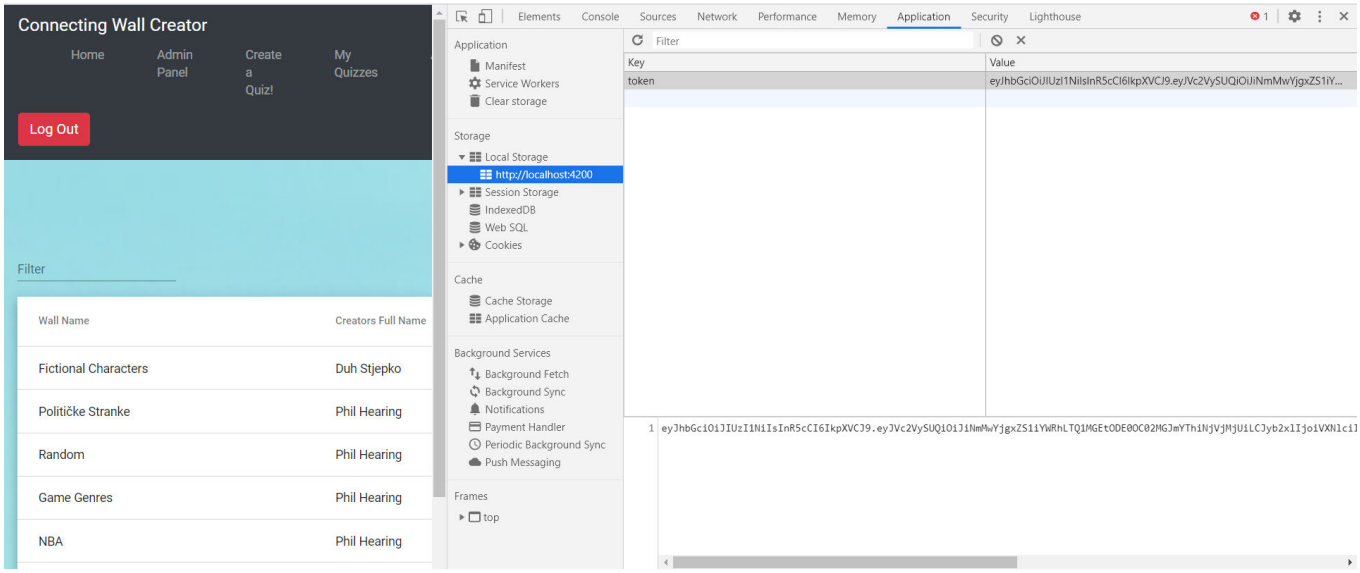
Slika 4.14. *Kod iz Startup dokumenta koji služi za JWT Autentikaciju.*

```
1.          [HttpPost]
2.          [Route("Login")]  //Post: api/ApplicationUser/Login
3.          public async Task<IActionResult> Login(LoginModel model)
4.          {
5.              var user = await _userManager.FindByNameAsync(model.UserName);
6.          if (user != null && await _userManager.CheckPasswordAsync(user, model.Password))
7.              {
8.              var role = await _userManager.GetRolesAsync(user);  //dohvati ulogu
    korisnika
9.                  IdentityOptions _options = new IdentityOptions();
10.                 var tokenDescriptor = new SecurityTokenDescriptor
11.                 {
12.                     Subject = new ClaimsIdentity(new Claim[]
13.                     {
14.                  new Claim ("UserID", user.Id.ToString()),
15.                 new Claim(_options.ClaimsIdentity.RoleClaimType,role.FirstOrDefault())}),
16.                     Expires = DateTime.UtcNow.AddDays(1),//token ističe za jedan dan
17.                     SigningCredentials = new SigningCredentials(new SymmetricSecurityKe
    y(Encoding.UTF8.GetBytes(_appSettings.JWT_Secret)), SecurityAlgorithms.HmacSha256Signat
    ure)//dohvaćanje tajnog niza znakova i određivanje algoritma enkripcije
18.                 };
19.                 var tokenHandler = new JwtSecurityTokenHandler();
20.                 var securityToken = tokenHandler.CreateToken(tokenDescriptor);
21.                 var token = tokenHandler.WriteToken(securityToken);
22.                 return Ok(new { token});
23.             }
24.             else
25.             {
26.                 return BadRequest(new { message = "Username or password is incorrect");
27.             }
28.         }
29.     }
```

Slika 4.15. *Kod za prijavu korisnika pomoću JWT Tokena.*



Slika 4.16. *Dohvaćanje korisničkih podataka prijavljenog korisnika po njegovom Tokenu u programu Postman.*



Slika 4.17. *Token trenutno prijavljenog korisnika unutar lokalne memorije web preglednika.*

4.2.5 Registracija korisnika po ulozi

Aplikacija „Zid za spajanje“ posjeduje 2 tipa uloga (*eng. Role*) - administrator i korisnik. Administratoru će biti dozvoljen pristup Admin Panel komponenti preko koje može brisati bilo kojeg korisnika i bilo koji napravljeni zid za spajanje, dok ostali korisnici neće imati tu privilegiju. Prvi korak kod odvajanja korisničkih uloga je dodati dvije n-torke u tablicu AspNetRoles (napravljena migracijom IdentityUser klase) s atributima Name (ime uloge) i Id (identifikator uloge). Nakon toga, u postojeću klasu ApplicationUser dodaje se novi atribut koji će predstavljati ulogu korisnika. Njegova će se vrijednost provjeravati s korisničke strane aplikacije, te će se ovisno o njegovoj vrijednosti dozvoliti ili zabraniti pristup Admin Panel komponenti. Nakon definiranja atributa uloga, potrebno je omogućiti dodavanje uloga novim korisnicima unutar Startup dokumenta kao i navođenje uloge novog korisnika unutar ApplicationUserController upravljača. Svi korisnici će imati ulogu „User“ osim jednog koji će imati „Admin“, a kreiranje novog admin računa (korisnika s Admin ulogom) neće biti omogućeno. Implementacija dohvaćanja korisničke uloge na klijentskoj strani i restrikcija pristupa ovisno o ulozi bit će objašnjena u poglavlju 4.3.

Results | Messages

	Id	Name	NormalizedName	ConcurrencyStamp
1	1	Admin	Admin	NULL
2	2	User	User	NULL

Slika 4.18. *Sadržaj tablice AspNetRoles.*

```
services.AddDefaultIdentity<Models.ApplicationUser>()
                .AddRoles<IdentityRole>()
                .AddEntityFrameworkStores<Models.AuthenticationContext>();
```

Slika 4.19. *Kod za dodavanje uloga za svaki napravljeni entitet Korisnik aplikacije.*

```
[HttpPost]
        [Route("Register")]
        //Post: api/ApplicationUser/Register

       public async Task<Object> PostApplicationUser(Models.ApplicationUserModel model)

        {
            model.Role = "User"; //Uloga registriranog korisnika
            var applicationUser = new Models.ApplicationUser()
            {
                UserName = model.Username,
                Email = model.Email,
                FullName = model.FullName
            };
        try
        {
          var result =await  _userManager.CreateAsync(applicationUser, model.Password);

          await _userManager.AddToRoleAsync(applicationUser, model.Role); //Dodavanje
   uloge korisniku u kontekst klasi
          return Ok(result);
         }
           catch (Exception ex)
           {

               throw ex;
           }
        }
```

Slika 4.20. *HTTP Post metoda za registraciju korisnika, te dodavanje korisniku ulogu „User“.*

4.2.6 Brisanje objekata u bazi

Aplikacija omogućuje korisnicima brisanje vlastitih zidova, a administratoru brisanje svih zidova i korisnika. Pošto svaki zid ima jednog korisnika koji ga je napravio, a svaki pojam i grupna veza točno jedan zid kojem pripadaju, potrebno je omogućiti kaskadno brisanje entiteta. Kaskadno brisanje (*eng. Cascade Delete*) je proces brisanja n-torki (unutar tablica) kod kojeg se brišu i one n-torke koje imaju vezu na n-torku koja se briše. Entity Framework okvir i SQL baza podataka neće dozvoliti brisanje korisnika ako je taj korisnik napravio zid, jer taj zid također sadrži vezu (strani ključ) na korisnika. Kaskadno brisanje bi u ovom primjeru trebalo obrisati sve zidove korisnika, pa i pojmove i grupne veze svih zidova. Kaskadno brisanje ostvarilo se stvaranjem okidača (*eng. Trigger*) na tablicama gdje su primarni ključevi n-torki (tablice roditelji, *eng. Parent Table*) strani ključevi n-torki u drugim tablicama (tablice djeca, *eng. Child Table*). Tablica AspNetUsers roditelj je tablici Walls, a tablica Walls roditelj je tablicama Terms i GroupConnections. Dakle, stvaraju se okidači nad tablicama AspNetUsers i Walls preko upita u SQL Server Management Studiu. Okidači se izvode umjesto operacije brisanje, a izvedeni su tako da rade unutarnje pridruživanje (*eng. Inner Join*) primarnih ključeva roditeljskih tablica i stranih ključeva tablica djece, te ih postavljaju u nulu (prazan niz znakova). Nakon pridruživanja, brišu se sve n-torke kojima je vrijednost primarnog ključa i stranog ključa jednaka praznom nizu znakova.

```
1.  CREATE TRIGGER Trigger_Users
2.      ON [dbo].AspNetUsers
3.      INSTEAD OF DELETE
4.      AS
5.      BEGIN
6.      SET NOCOUNT ON;
7.      UPDATE Walls
8.      SET Walls.userID=null
9.      from [dbo].Walls Walls inner join
10.     deleted d on d.Id=Walls.userID
11.     UPDATE Terms
12.     set Terms.wallID=null
13.     from [dbo].Walls Walls inner join
14.     deleted d on d.Id=Walls.userID
15.     UPDATE Terms
16.     set Terms.wallID1=null
17.     from [dbo].Walls Walls inner join
18.     deleted d on d.Id=Walls.userID
19.     UPDATE Terms
20.     set Terms.wallID2=null
21.     from [dbo].Walls Walls inner join
22.     deleted d on d.Id=Walls.userID
23.     UPDATE Terms
24.     set Terms.wallID3=null
25.     from [dbo].Walls Walls inner join
26.     deleted d on d.Id=Walls.userID
27.     UPDATE GroupConnections
```

```
    set GroupConnections.wallID=null
    from [dbo].Walls Walls inner join
    deleted d on d.Id=Walls.userID
    UPDATE GroupConnections
    set GroupConnections.wallID1=null
    from [dbo].Walls Walls inner join
    deleted d on d.Id=Walls.userID
    UPDATE GroupConnections
    set GroupConnections.wallID2=null
    from [dbo].Walls Walls inner join
    deleted d on d.Id=Walls.userID
    UPDATE GroupConnections
    set GroupConnections.wallID3=null
    from [dbo].Walls Walls inner join
    deleted d on d.Id=Walls.userID
    end;
    delete from AspNetUsers where Id in (SELECT Id FROM deleted)
    delete from Walls where (userID is null)
```

Slika 4.21. *SQL upit za stvaranje okidača (eng.Trigger) na tablici AspNetUsers (korisnik) koji omogućuje kaskadno brisanje n-torki tablice.*

```
CREATE TRIGGER TRIGGER_Walls
    ON [dbo].Walls
    INSTEAD OF DELETE
    AS
    BEGIN
    SET NOCOUNT ON;
    UPDATE Terms
    SET Terms.wallID = null
    from [dbo].Terms Terms inner join
    deleted d on d.wallID=Terms.wallID
    UPDATE Terms
    SET Terms.wallID1 = null
    from [dbo].Terms Terms inner join
    deleted d on d.wallID=Terms.wallID1
    UPDATE Terms
    SET Terms.wallID2 = null
    from [dbo].Terms Terms inner join
    deleted d on d.wallID=Terms.wallID2
    UPDATE Terms
    SET Terms.wallID3 = null
    from [dbo].Terms Terms inner join
    deleted d on d.wallID=Terms.wallID3
    UPDATE GroupConnections
    SET GroupConnections.wallID=null
    from [dbo].GroupConnections GroupConnections inner join
    deleted g on g.wallID=GroupConnections.wallID
    UPDATE GroupConnections
    SET GroupConnections.wallID1=null
    from [dbo].GroupConnections GroupConnections inner join
    deleted g on g.wallID=GroupConnections.wallID1
    UPDATE GroupConnections
    SET GroupConnections.wallID2=null
    from [dbo].GroupConnections GroupConnections inner join
    deleted g on g.wallID=GroupConnections.wallID2
    UPDATE GroupConnections
    SET GroupConnections.wallID3=null
    from [dbo].GroupConnections GroupConnections inner join
    deleted g on g.wallID=GroupConnections.wallID3
    END;
    delete from Walls where wallID in (select wallID from deleted)
    delete from Terms where (wallID IS NULL and wallID1 IS NULL and wallID2 IS NULL and
  wallID3 IS NULL)
```

```
42.     delete from GroupConnections where (wallID IS NULL and wallID1 IS NULL and wallID2
    IS NULL and wallID3 IS NULL)
43.     GO
```

Slika 4.22. *SQL upit za stvaranje okidača na tablici Walls (zid za spajanje) koji omogućuje kaskadno brisanje n-torki tablice.*

4.3 Izrada korisničkog sučelja

U ovom poglavlju će se opisati izrada web aplikacije u Angular 9 okviru. Opisat će se struktura aplikacije (podjela u komponente), način upisivanja podataka u bazu preko Angular formi (registracija korisnika, prijava korisnika i objavljivanje vlastitog kviza), dohvaćanje korisnika i zidova, te njihova prezentacija u Angular Material tablicama. Nadalje, opisat će se modifikacija zidova i igranje zida za spajanje preko Angular skočnih prozora (*eng. Angular Popup Dialog*) kao i restrikcija pristupa Admin Panel komponenti. Naposljetku, opisat će se stvaranje igrivog kviza „Zid za spajanje“.

4.3.1 Podjela u komponente

Kao što je opisano u poglavlju 3.5, Angular aplikacija sastoji se od više odvojenih komponenti, gdje svaka komponenta funkcionira kao zasebna jedinica, to jest, ima svoj HTML, CSS i TypeScript dokument. HTML datoteka prikazuje sadržaj, CSS stilizira sadržaj, a TypeScript datoteka opisuje ponašanje sadržaja i omogućuje upravljanje podacima.

Aplikacija Zid za spajanje sastoji se od sljedećih komponenti (abecedno) :

AboutComponent – komponenta koja sadrži podatke o aplikaciji i kontakt programera aplikacije,

AdminPanelComponent – komponenta kojoj pristup ima samo korisnik s ulogom „Admin“. Prikazuje sve registrirane korisnike s ulogom „User“, te sve napravljene zidove za spajanje (kvizove). Implementirana je mogućnost brisanja korisnika i zida pozivom HTTP delete metodi iz upravljača s poslužiteljske strane.

CreateQuizComponent – sadrži formu za stvaranje spajajućeg zida. Njezina TypeScript datoteka sadrži HTTP Post metodu koja služi za upisivanje podataka iz forme u bazu podataka. Podaci iz forme dohvaćeni su i pretvoreni u tijelo zahtjeva za upis podataka u bazu na način koji

propisuje upravljač poslužiteljske strane WallController. Kod upisivanja podataka u formu nisu dozvoljena prazna polja, što je implementirano korištenjem Angular validatora formi.

EditWallComponent – komponenta za izmjenu podataka pojedinog zida. Otvara se kao skočni prozor unutar MyQuizzes komponente. Sastoji se od forme koja je identična formi za stvaranje zida, ali su podaci forme popunjeni podacima onog kviza koji je odabran na MyQuizzes komponenti. Korisnik unutar nje može promijeniti postojeći kviz (implementacija HTTP Put metode).

ForbiddenComponent – komponenta se prikazuje ako prijavljeni korisnik s ulogom „User“ želi pristupiti AdminPanel komponenti.

GameComponent – komponenta koja sadrži implementaciju igre „Zid za spajanje“. Sastoji se od brojnih HTML elemenata koji se prikazuju i sakrivaju u ovisnosti o događajima u TypeScript datoteci.

HeaderComponent – navigacijska traka implementirana kao zasebna komponenta. Preko routerLink funkcije koja zahtjeva navođenje svih ruta u aplikaciji (unutar app-routing.module TypeScript datoteke) u navigacijskoj se traci prelazi s jedne rute (komponente) na drugu. Prisutna je u svim glavnim navigacijskim komponentama (Home, Admin Panel, CreateQuiz, MyQuizzes i About).

HomeComponent – u njoj je prikazana tablica sa svim napravljenim kvizovima. Tablica omogućuje pretraživanje kviza po imenu, te odabir broja kvizova koji će biti prikazani. Klikom na gumb PlayQuiz otvara se skočni prozor s otvorenom PlayQuiz komponentom. Pri otvaranju svakog zida implementirano je slanje identifikatora odabranog zida u komponentu koja se otvara u skočnom prozoru.

LoginComponent – komponenta za prijavljivanje korisnika. Ako korisnik postoji, te ako je prijava uspješna, korisnik je preusmjeren na Home komponentu.

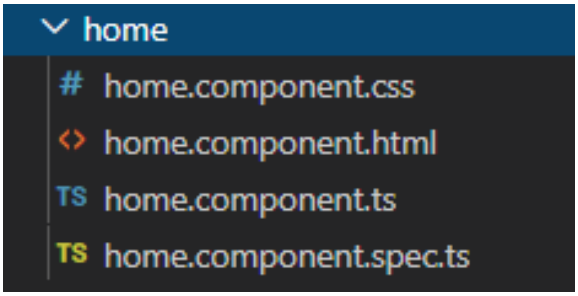
MyQuizzesComponent – prikazuje sve kvizove koje je napravio trenutno prijavljeni korisnik. Korisnik može promijeniti sve pojmove, grupne veze i imena napravljenih zidova, kao i obrisati zidove. Unutar TypeScript datoteke ove komponente implementiran je HTTP zahtjev za dohvaćanje svih zidova po korisničkom identifikatoru.

PlayQuizComponent – komponenta koja se otvara u skočnom prozoru kad korisnik odabire zid za spajanje koji želi igrati na Home komponenti. Prima identifikator zida za spajanje (s Home

komponente) i prikazuje ime zida za spajanje, te korisnika koji ga je napravio. Sadrži opis pravila kviza. Klikom na gumb Start korisniku prikazuje QuizStart komponentu.

RegistrationComponent – komponenta za registraciju korisnika. Sastoji se od forme u koju je nužno unijeti korisničko ime, e-mail, puno ime, lozinku i potvrđenu lozinku. U slučaju neispravnog unosa, pojavljuje se poruka o neispravnom unosu i onemogućuje se registracija, što je implementirano pomoću validatora.

QuizStartComponent – komponenta koja se otvara unutar skočnog prozora nakon što korisnik pritisne gumb Start na PlayQuiz komponenti. Unutar nje se nalazi Game komponenta. Od ostalih komponenti koristit će se dva servis dokumenta (wall.service i user.service) koji sadrže HTTP zahtjeve i strukturu za stvaranje, dohvaćanje, izmjenu i brisanje podataka. Podaci dobiveni iz metoda servisa mogu se koristiti u svakoj komponenti ako se prethodno instancira objekt servis klase.



Slika 4.23. *Prikaz strukture komponente u Angular 2+ okviru.*

```
<app-header></app-header>
<div class="container">
<div class="d-flex justify-content-center" *ngIf="userDetails"><!--moram upisat ngif inace baca error-->
<h1 class="display-1">Welcome <strong>{{userDetails.userName}} !</strong></h1>
</div>
<div class="d-flex justify-content-center">
<p class="h2">Ready to play?</p></div>
<div class="search-div">
    <mat-form-field>
        <input matInput (keyup)="applyFilter($event.target.value)" placeholder="Filter" autocomplete="off">
    </mat-form-field>
</div>
<div class="example-container mat-elevation-z8">
    <mat-table [dataSource]="dataSource" matSort>
        <!--Column ID-->
        <ng-container matColumnDef="WallName">
            <mat-header-cell *matHeaderCellDef mat-sort-header>Wall Name</mat-header-cell>
            <mat-cell *matCellDef="let row">{{row.wallName}}</mat-cell>
        </ng-container>
        <ng-container matColumnDef="Creator FullName">
            <mat-header-cell *matHeaderCellDef mat-sort-header>Creators Full Name</mat-header-cell>
            <mat-cell *matCellDef="let row">{{row.user.fullName}}</mat-cell>
        </ng-container>
        <ng-container matColumnDef="Creator UserName">
            <mat-header-cell *matHeaderCellDef mat-sort-header>Creators Username</mat-header-cell>
```

Slika 4.24. *Prikaz uključivanja Header komponente (<app-header> oznaka) u Home komponentu koristeći odgovarajući birač (eng. Selector).*

4.3.2 Registracija i prijava korisnika

Za registraciju korisnika potrebno je napraviti formu u koju korisnik unosi svoje korisničko ime, e-mail adresu, puno ime, lozinku i potvrđenu lozinku. Za stiliziranje forme koristile su se gotove klase Bootstrap alata form-group i form-row. Podaci koji se unose u formu bit će vezani za podatke unutar TypeScript datoteke user.service. Servis datoteke sadrže funkcije koje se pozivaju u komponentama, a najčešće su to HTTP zahtjevi prema poslužiteljskoj strani. Datoteka user.service sastoji se od tijela zahtjeva za registraciju korisnika, tijela forme za registraciju, funkcija za prijavu korisnika, te funkcija za brisanje i dohvaćanje podataka o korisniku. Za registraciju korisnika potrebno je dohvatiti unesene podatke iz forme, spremiti ih u tijelo zahtjeva, te napraviti HTTP Post zahtjev iz Registration komponente kad korisnik stisne SignUp! gumb. Kod popunjavanja forme definirani su validatori koji onemogućuju unos praznog niza znakova i neispravan unos e-mail adrese (unos mora sadržavati znak '@'). Nakon slanja HTTP Post zahtjeva prema poslužiteljskoj strani dohvaća se odgovor na zahtjev kojeg šalje poslužitelj (pokrenuti Web API) preko kojeg se može vidjeti je li registracija korisnika bila uspješna. U ovisnosti o vrijednosti zahtjeva prikazuje se poruka o uspješnoj ili neuspješnoj registraciji. Poruka koja se šalje prikazana je pomoću Toastr biblioteke koja služi za transparentan prikaz obavijesti na web stranicama. Prijava korisnika analogna je registraciji, unutar komponente

Login poziva se HTTP Post funkcija iz user.service datoteke. Ako je zahtjev uspješan, Token prijavljenog korisnika sprema se u lokalnu memoriju preglednika i korisnik se navigira na Home komponentu. Ako je status odgovora jednak 400 (loš zahtjev), korisniku se prikazuje poruka o neispravnom zahtjevu.

```
14      constructor(private fb:FormBuilder, private http:HttpClient) {
15      }
16      //adresa API-a
17      readonly BaseURI = 'http://localhost:57392/api';
18
19      formModel=this.fb.group({
20        UserName : ['',Validators.required],
21        Email : ['',[Validators.required,Validators.email]],
22        FullName : ['',Validators.required],
23        Password : ['',[Validators.required, Validators.minLength(6)]],
24        ConfirmedPassword : ['',Validators.required]},
25        {validator : this.comparePasswords});
26
27      comparePasswords(fb :FormGroup){
28        let confirmPswrdCtrl=fb.get('ConfirmedPassword');
29        if(confirmPswrdCtrl.errors==null || 'passwordMismatch' in confirmPswrdCtrl.errors){
30          if(fb.get('Password').value!=confirmPswrdCtrl.value)
31          {
32          confirmPswrdCtrl.setErrors({passwordMismatch:true});
33          }
34         else
35         {
36          confirmPswrdCtrl.setErrors(null);
37         }
38      }
39    }
```

Slika 4.25. *Prikaz definiranja podataka od kojih će se sastojati forma za registraciju, te funkcija za provjeru jednakosti lozinke i potvrđene lozinke.*

```
42  //HTTP Post
43  register(){
44   var body={
45    UserName:this.formModel.value.UserName,
46    Email : this.formModel.value.Email,
47    FullName : this.formModel.value.FullName,
48    Password : this.formModel.value.Password,
49   };
50   return this.http.post(this.BaseURI+'/ApplicationUser/Register', body);
51  }
52
53  //
54  login(formData)
55  {
56    return this.http.post(this.BaseURI+'/ApplicationUser/Login', formData);
57  }
58
59  getUserProfile()
60  {
61    var tokenHeader = new HttpHeaders({'Authorization':'Bearer ' + localStorage.getItem('token')})
62    return this.http.get(this.BaseURI+'/UserProfile', {headers:tokenHeader});
63  }
64
```

Slika 4.26. *Definiranje tijela zahtjeva za registraciju, funkcije za prijavu korisnika, te funkcije za dohvaćanje podataka o prijavljenom korisniku.*

```
export class RegistrationComponent implements OnInit {

  constructor(public service : UserService, private toastr:ToastrService) { }

  ngOnInit(): void {
    this.service.formModel.reset();  //kod pokretanja komponente forma je prazna
  }
  //na click gumba
  onSubmit(){
    this.service.register().subscribe(  //na poziv funkcije iz user.service datoteke
      (res:any)=>{
        if(res.succeeded){  //ako je zahtjev uspješan
          this.service.formModel.reset();  //ukloni podatke iz forme
          this.toastr.success('New user registered!', 'Registration successful.');
        }  //Toastr poruka o uspješnoj operaciji
        else{
          res.errors.forEach(element => {
            switch(element.code){
              case 'DuplicateUserName':  //ako već postoji korisničko ime
                //username taken
                this.toastr.error('Username is already taken!','Registration failed.');
       //poruka o postojećem korisničkom imenu
                break;
                default:
                this.toastr.error(element.description,'Registration failed.')
                //Poruka ako je druga greška u pitanju
                break;
            }
          });
        }
      },
      err=>{//Poruka o greški upisuje se u konzolu preglednika
        console.log(err);
      }
    );
  }
}
```

Slika 4.27. *TypeSript datoteka Registration komponente.*

```
constructor(private service:UserService, private router:Router, private toastr:ToastrSe
rvice) { }

  ngOnInit(): void {
    if(localStorage.getItem('token')!=null)//ako postoji token
    {
      this.router.navigateByUrl('/home');
    }
  }
  onSubmit(form:NgForm)
  {
    this.service.login(form.value).subscribe(
      (res:any)=>{//ako login uspije
        localStorage.setItem('token',res.token);//token se stavlja u lokalnu memoriju
        this.router.navigateByUrl('/home');
      },
      err=>{
        if(err.status==400)//korisnik ne postoji
        {
          this.toastr.error('Incorrect Username or Password.', 'Authentication Failed')
;
        }
        else{
```

```
22.           console.log(err);
23.         }
24.       }
25.     );
26.   }
27. }
```

Slika 4.28. *Prijava korisnika unutar Login komponente.*

4.3.3 Restrikcija pristupa Admin Panel komponenti

Pošto je u aplikaciji omogućena registracija korisnika po ulozi Admin i User, na korisničkom sučelju potrebno je dohvatiti podatak o ulozi prijavljenog korisnika, te ovisno o vrijednosti tog podatka omogućiti ili onemogućiti pristup komponenti Admin Panel. Sadržaj komponente opisan je u poglavlju 4.3.1. Za omogućavanje funkcionalnosti restrikcije pristupa stvorena je nova TypeScript datoteka auth.guard koja sadrži funkciju canActivate. Funkcija canActivate ispituje postoji li token u lokalnoj memoriji (je li korisnik prijavljen), te ispituje ulogu prijavljenog korisnika preko funkcije iz user.service datoteke roleMatch. Funkcija roleMatch prima dozvoljenu ulogu i uspoređuje ju s ulogom dobivenom iz Tokena prijavljenog korisnika. Ako je prijavljeni korisnik ima ulogu Admin (navedena kod pristupanja ruti Admin Panel u app-routing-modulu), canActivate funkcija vraća logičku istinu i komponenta kojoj se pristupa se može otvoriti. Ako prijavljeni korisnik ne posjeduje odgovarajuću ulogu, preusmjerava se na Forbidden komponentu.

```
const routes: Routes = [
  {path:'', redirectTo:'/user/login',pathMatch:'full'},//first component when app starts
  {path:'user', component:UserComponent,
  children: [
    {path:'registration', component:RegistrationComponent},
    {path: 'login', component:LoginComponent}
  ]
},
  {path : 'home', component:HomeComponent,
  children:[
    {path:'quizstart',component:QuizStartComponent}
  ]
}, // for disabling access to unlogged user
  {path : 'forbidden', component:ForbiddenComponent},//public - no auth guard
  {path : 'adminpanel', component:AdminPanelComponent, canActivate:[AuthGuard], data:{permittedRoles:['Admin']} },
  {path : 'create-quiz', component: CreateQuizComponent,canActivate:[AuthGuard]},
  {path : 'my-quizzes', component:MyQuizzesComponent,canActivate:[AuthGuard]},
  {path: 'about', component:AboutComponent,canActivate:[AuthGuard]}

];
```

Slika 4.29. *Navođenje ruta u aplikaciji unutar app-routing-modulea. Kod rute za pristup Admin Panel komponenti navedena je dozvoljena uloga.*

```
canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): boolean{
      if(localStorage.getItem('token')!=null){  //ako token postoji
      let roles = next.data['permittedRoles'] as Array<string>;//uloge kojima je dopušten pristup
      if(roles)
      {
        if(this.service.roleMatch(roles)) return true;  //ako funkcija roleMatch vraća istinu
        else{
          this.router.navigate(['forbidden']); //ako funkcija roleMatch ne vraća istinu
          return false;
        }
      }
      return true;
    }
    else
    {
      this.router.navigate(['/user/login']);  //ako token ne postoji
      return false;
    }
  }
```

Slika 4.30. *Prikaz canActivate funkcije čiji rezultat ovisi o prikazivanju Admin Panel komponente.*

```
roleMatch(allowedRoles): boolean
{
  var isMatch=false;
  var payload = JSON.parse(window.atob(localStorage.getItem('token').split('.')[1]));
  var userRole=payload.role;
  allowedRoles.forEach(element => {
    if(userRole==element)
    {
      isMatch=true;
      return false;

    }
  });
  return isMatch;
}
```

Slika 4.31. *Prikaz roleMatch funkcije iz user.service datoteke. Ona se poziva u funkciji canActivate i vraća ulogu prijavljenog korisnika iz tijela njegovog Tokena.*

Slika 4.32. *Komponenta na koju se preusmjerava korisnik s ulogom User kad pokuša pristupiti Admin Panel komponenti.*

4.3.4 Unos „Zida za spajanje“

Korisniku aplikacije potrebno je omogućiti unos vlastitog zida preko Angular formi. Kao kod unosa podataka za prijavu i registraciju, unutar servis dokumenta potrebno je definirati model forme za zid koji će se unositi. Taj model sastoji se od podataka : ime zida, grupi formi koje predstavljaju pojmove grupe A, B, C i D, te grupnih veza pojmova. Validatori unutar definicije forme onemogućuju unos praznog niza znakova za ime pojma, te zahtijevaju unos od minimalno 3 znaka za opis veze između pojmova u grupi. Nadalje, potrebno je formatirati HTTP Post zahtjev dohvaćanjem podataka iz forme za unos zida. U tijelu zahtjeva nužno je proslijediti identifikator korisnika koji kreira zid. Zbog toga je u CreateQuiz komponenti potrebno instancirati i user.service klasu, te preko njezine funkcije getUserProfile dohvatiti identifikator prijavljenog korisnika, te ga proslijediti zajedno s ostalim podacima o zidu koji se unosi. Time je ispoštovan relacijski model podataka koji zahtjeva da svaki zid ima jednog korisnika, te da jedan korisnik može unijeti više zidova.

```
CreateQuiz()
  {
var body={
   wallName:this.wallService.formModel.value.wallName,
   userID:this.userDetails.id,//id trenutno logiranog usera
   groupATerms:[
    {termName: this.wallService.formModel.get("GroupATerms").get("GroupATerm1").value},
    {termName: this.wallService.formModel.get("GroupATerms").get("GroupATerm2").value},
    {termName: this.wallService.formModel.get("GroupATerms").get("GroupATerm3").value},
    {termName: this.wallService.formModel.get("GroupATerms").get("GroupATerm4").value},
   groupBTerms:[
    {termName: this.wallService.formModel.get("GroupBTerms").get("GroupBTerm1").value},
    {termName: this.wallService.formModel.get("GroupBTerms").get("GroupBTerm2").value},
    {termName: this.wallService.formModel.get("GroupBTerms").get("GroupBTerm3").value},
    {termName: this.wallService.formModel.get("GroupBTerms").get("GroupBTerm4").value},
   groupCTerms:[
    {termName: this.wallService.formModel.get("GroupCTerms").get("GroupCTerm1").value},
    {termName: this.wallService.formModel.get("GroupCTerms").get("GroupCTerm2").value},
    {termName: this.wallService.formModel.get("GroupCTerms").get("GroupCTerm3").value},
    {termName: this.wallService.formModel.get("GroupCTerms").get("GroupCTerm4").value},
   groupDTerms:[
    {termName: this.wallService.formModel.get("GroupDTerms").get("GroupDTerm1").value},
    {termName: this.wallService.formModel.get("GroupDTerms").get("GroupDTerm2").value},
    {termName: this.wallService.formModel.get("GroupDTerms").get("GroupDTerm3").value},
    {termName: this.wallService.formModel.get("GroupDTerms").get("GroupDTerm4").value},
        groupAConnections:[
          {connectionName:this.wallService.formModel.value.GroupAConnections}
        ]
        ,
        groupBConnections:[
          {connectionName:this.wallService.formModel.value.GroupBConnections}
        ]
        ,
        groupCConnections:[
          {connectionName:this.wallService.formModel.value.GroupCConnections}
        ]
        ,
        groupDConnections:[
          {connectionName:this.wallService.formModel.value.GroupDConnections}
        ]
    }
    return this.http.post(this.BaseURI+'/Wall', body,{headers:this.header});
  }
```

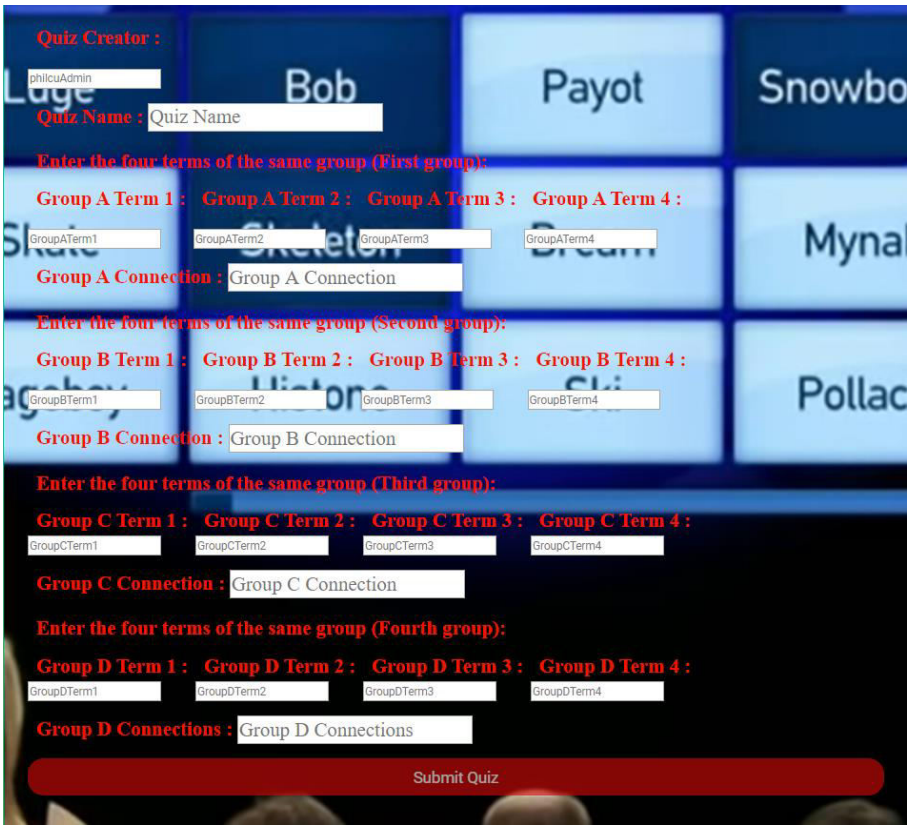
Slika 4.33. *HTTP zahtjev za unošenje zida s dohvaćanjem podataka iz forme za unos zida. Identifikator zida dohvaća se iz objekta UserDetails.*

```
22    ngOnInit(): void {
23      this.wallService.formModel.reset();
24      this.service.getUserProfile().subscribe(
25        res=>{
26          this.userDetails=res;
27        },
28        err=>{
29          console.log(err);
30        }
31      );
32    }
```

Slika 4.34. *Dohvaćanje podataka o trenutno prijavljenom korisniku unutar CreateQuiz komponente. Spremanje podataka u objekt WallDetails.*

```
87    onSubmit(){
88
89      this.CreateQuiz().subscribe(
90        (result)=>{
91          this.toastr.success('New Connecting Wall Created!','Wall Creation Successful! Go to My Quizzes to edit it if necessary!');
92          console.log("result",result);
93        }
94      )
95      this.wallService.formModel.reset();
96    }
97  }
```

Slika 4.35. *Funkcija onSubmit koja se pokreće na klik gumba. U slučaju uspješnog zahtjeva, prikazana je poruka o uspješnom unosu zida.*



Slika 4.36. *Prikaz forme za unos zida na CreateQuiz komponenti.*

4.3.5 Dohvaćanje, prikaz i brisanje podataka

Kod dohvaćanja podataka definirane su funkcije (HTTP zahtjevi) unutar servis datoteka. Postoje dva servis dokumenta – wall.service i user.service, a oba su zadužena za HTTP operacije unosa,dohvaćanja, izmjene i brisanja zidova za spajanje i korisnika. Dohvaćanje svih zidova obavit će se unutar komponenti AdminPanel i Home, a dohvaćanje zidova po korisničkom identifikatoru unutar MyQuizzes komponente. Unutar Admin Panel komponente bit će dohvaćeni i svi korisnici s ulogom „User“. Svi dohvaćeni podaci (objekti Zid i Korisnik) prikazat će se pomoću Angular Material podatkovne tablice (*eng. Data Table*), strukture koju omogućuje službena Angular biblioteka MatTableModule. Kod definiranja prikaza unutar tablice potrebno je navesti izvor podataka koje tablica prikazuje, a podaci moraju biti MatTableDataSource tipa. Zbog toga će se nakon dohvaćanja objekata (zid ili korisnik) morati raditi njihova pretvorba u MatTableDataSource objekte. Uz definiranje izvora podataka, potrebno je navesti stupce tablice unutar kojih će svaki atribut objekta biti prikazan. Nadalje, omogućit će se pretraga postojećih korisnika i zidova preko unosa u tekstualno polje. Tablici će se dodati i MatPaginator struktura preko koje će se elementi tablice sortirati u više stranica. Unutar zaglavlja tablice dodat će se dva reda (tipa mat-footer-row) koja će prikazivati odgovarajuću poruku ako objekti tablice još nisu dohvaćeni ili ako ne postoje podaci u tablici.

```
  this.wallService.getAllWalls().subscribe(
    res=>{
      this.walls=res;
      this.dataSource=new MatTableDataSource(this.walls);
      this.dataSource.paginator=this.quizPaginator;
      console.log(this.walls);
    }
  )
}
```

Slika 4.37. *Dohvaćanje svih napravljenih zidova, njihova pretvorba u MatTableDataSource tip za prikaz u tablici, te dodavanje MatPaginatora.*

```
<mat-table [dataSource]="dataSource" matSort>
        <!—Stupci tablice-->
  <ng-container matColumnDef="WallName">
     <mat-header-cell *matHeaderCellDef mat-sort-header>Wall Name</mat-header-cell>
         <mat-cell *matCellDef="let row">{{row.wallName}}</mat-cell>
   </ng-container>
  <ng-container matColumnDef="Creator FullName">
 <mat-header-cell *matHeaderCellDef mat-sort-header>Creators Full Name</mat-header-cell>
         <mat-cell *matCellDef="let row">{{row.user.fullName}}</mat-cell>
  </ng-container>
   <ng-container matColumnDef="Creator UserName">
            <mat-header-cell *matHeaderCellDef mat-sort-header>Creators Username</mat-header-cell>
            <mat-cell *matCellDef="let row">{{row.user.userName}}</mat-cell>
        </ng-container>
        <ng-container matColumnDef="Email">
            <mat-header-cell *matHeaderCellDef mat-sort-header>Creators E-Mail</mat-header-cell>
            <mat-cell *matCellDef="let row">{{row.user.email}}</mat-cell>
        </ng-container>
        <ng-container matColumnDef="Date Created">
            <mat-header-cell *matHeaderCellDef mat-sort-header>Date Created</mat-header-cell>
            <mat-cell *matCellDef="let row">{{row.dateCreated}}</mat-cell>
        </ng-container>
        <ng-container matColumnDef="Actions">
             <mat-header-cell *matHeaderCellDef mat-sort-header>Actions</mat-header-cell>
                <mat-cell *matCellDef="let row">
                <button type="button" class="btn btn-success" (click)="openPlayWindow(row.wallID)">Play Quiz</button>
                </mat-cell>
        </ng-container>
        <ng-container matColumnDef="loading">  <!—Ako se zidovi učitavaju-->
            <mat-footer-cell *matFooterCellDef colspan="5">
                Loading Walls...
            </mat-footer-cell>
        </ng-container>
        <ng-container matColumnDef="noData">  <!—ako tablica nema podataka-->
            <mat-footer-cell *matFooterCellDef colspan="5">
                No Walls Created
            </mat-footer-cell>
        </ng-container>
        <mat-header-row *matHeaderRowDef="columns"></mat-header-row>
        <mat-row *matRowDef="let row; columns: columns;"></mat-row>
       <mat-footer-row *matFooterRowDef="['loading']" [ngClass]="{'hide':dataSource!=null}"></mat-footer-row>  <!—sakrij ako postoje podaci -->
        <mat-footer-row *matFooterRowDef="['noData']" [ngClass]="{'hide':!(dataSource!=null && dataSource.data.length==0)}"></mat-footer-row>  <!—sakrij ako postoje podaci i ako polje podataka ima elemente -->
    </mat-table>
       <mat-paginator [pageSizeOptions]="[5, 10, 25, 100]" [pageSize]="5"showFirstLastButtons></mat-paginator>
```

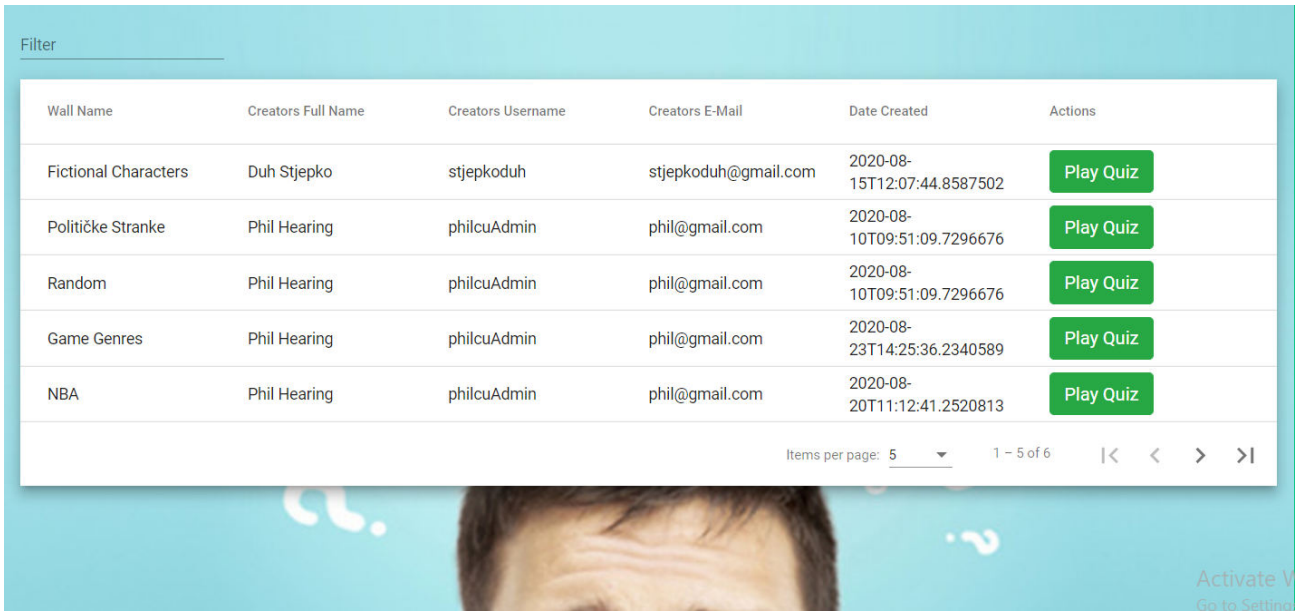
Slika 4.38. *Definiranje Angular Material tablice u HTML dijelu komponente, te navođenje njenog izvora podataka.*

```
<div class="search-div">
    <mat-form-field>
        <input matInput (keyup)="applyFilter($event.target.value)" placeholder="Filter" autocomplete="off">
    </mat-form-field>
</div>
```

Slika 4.39. *Filter podataka u tablici preko unosa u tekstualno polje.*

```
applyFilter(filterValue:string){
  filterValue=filterValue.trim();//whitespace removal
  filterValue=filterValue.toLowerCase();//datasource lowercase by default
  this.dataSource.filter=filterValue;
}
```

Slika 4.40. *Funkcija applyFilter koja prima unešenu vrijednost iz tekstualnog polja i filtrira dataSource polje po vrijednosti iz prvog stupca (ime zida ili ime korisnika).*



Slika 4.41. *Prikaz zidova dohvaćenih iz baze podataka, prikaz tekstualnog polja za pretragu zidova po korisničkom imenu, te zaglavlja za navigaciju između napravljenih zidova.*

Komponenta Admin Panel sadrži tablicu s gumbom za brisanje korisnika i zidova, dok MyQuizzes komponenta sadrži tablicu s gumbovima za brisanje i izmjenu postojećih zidova. Brisanje određenog zida za spajanje ostvaruje se pozivom funkcije deleteWall iz wall.service klase gdje se kao parametar predaje identifikator zida u redu Angular Material tablice gdje se

gumb Delete pritisnuo. Brisanje korisnika na Admin Panel komponenti ostvaruje se analogno brisanju zidova, to jest pozivom funkcije deleteUser koja prima identifikator korisnika.

```
ngOnInit(): void {
  this.userService.getUserProfile().pipe(
    switchMap(res => {
      this.userDetails = res;
      this.userID = this.userDetails.id;
      return this.service.getQuizByUserId(this.userID)
    })
  ).subscribe(res => {
      this.walls = res;
      this.dataSource=new MatTableDataSource(this.walls);
      this.dataSource.paginator=this.paginator;
      console.log(res);
  });

}
```

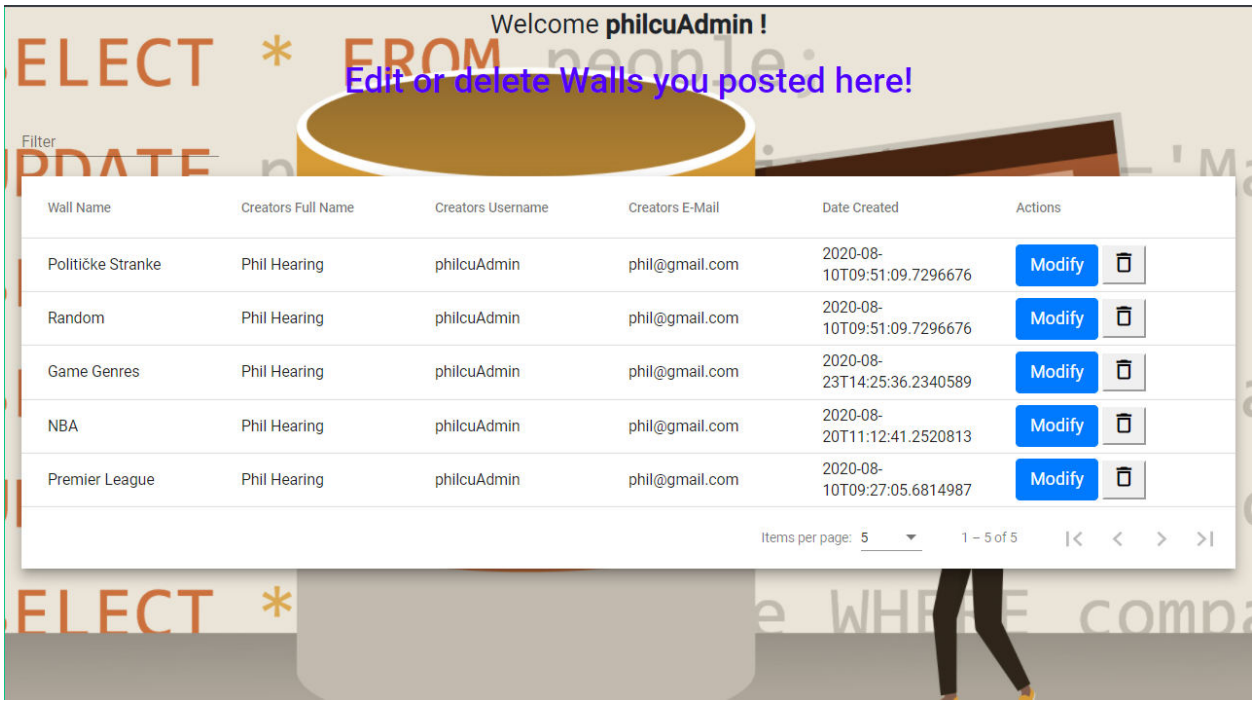
Slika 4.42. *Dohvaćanje identifikatora prijavljenog korisnika, te prosljeđivanje tog identifikatora u HTTP zahtjevu za dohvaćanje zidova po korisničkom identifikatoru. Navođenje podataka dobijenih zidova kao izvor za Angular Material podatkovnu tablicu.*

```
<ng-container matColumnDef="Actions">
    <mat-header-cell *matHeaderCellDef mat-sort-header>Actions</mat-header-cell>
       <mat-cell *matCellDef="let row">
       <button type="button" class="btn btn-primary" (click)="editWall(row.wallID)">Modify</button>
         <!--horizontal space-->
       <button mat-icon-button color="warn" (click)="DeleteWall(row.wallID)"><mat-icon>delete_outline</mat-icon></button>
       </mat-cell>
</ng-container>
```

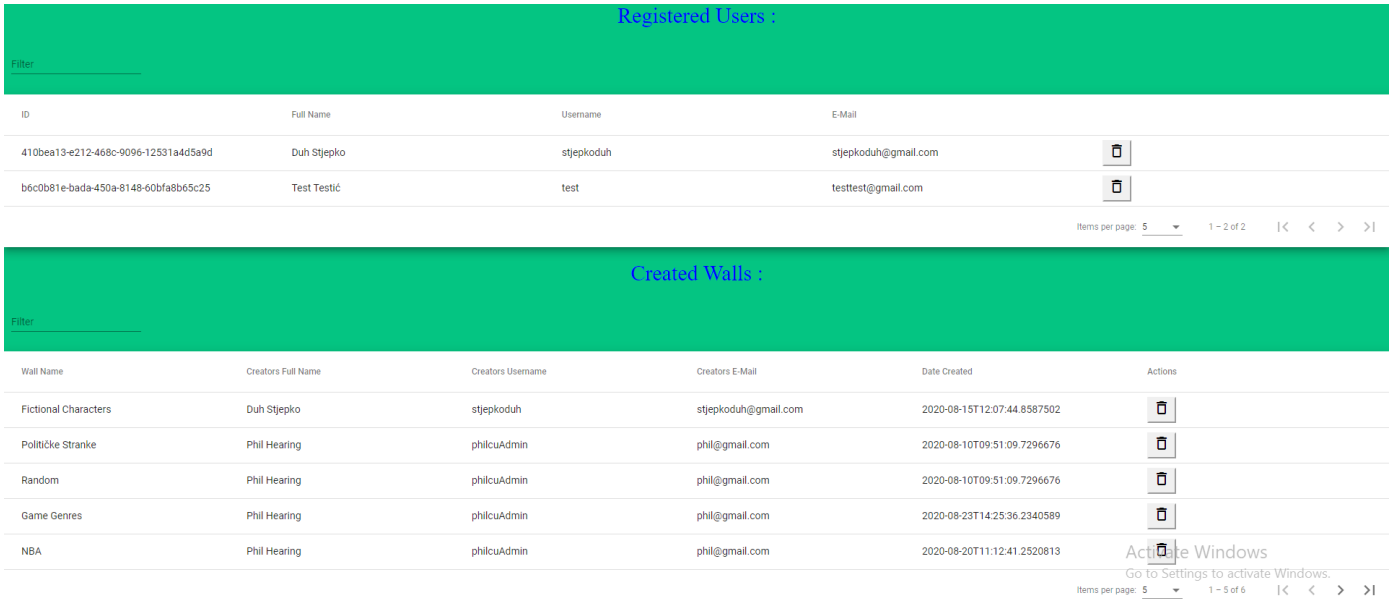
Slika 4.43. *Stupac Angular Material podatkovne tablice sa gumbovima za modificiranje i brisanje postojećih zidova unutar MyQuizzes komponente. Na događaj pristiska gumba pozivaju se funkcije editWall i DeleteWall s prosljeđenim identifikatorom zida koji je odabran.*

```
DeleteWall(wallId:string):void{
  if(confirm("Are you sure you want to delete this wall?")){
    this.service.deleteWall(wallId).subscribe(_ =>{
      this.walls=this.walls.filter(eachWall=>eachWall.wallID!==wallId);
      window.location.reload();
      this.toastr.success('Wall Deleted!','Wall Deletion Successful!');
    });
  }
}
```

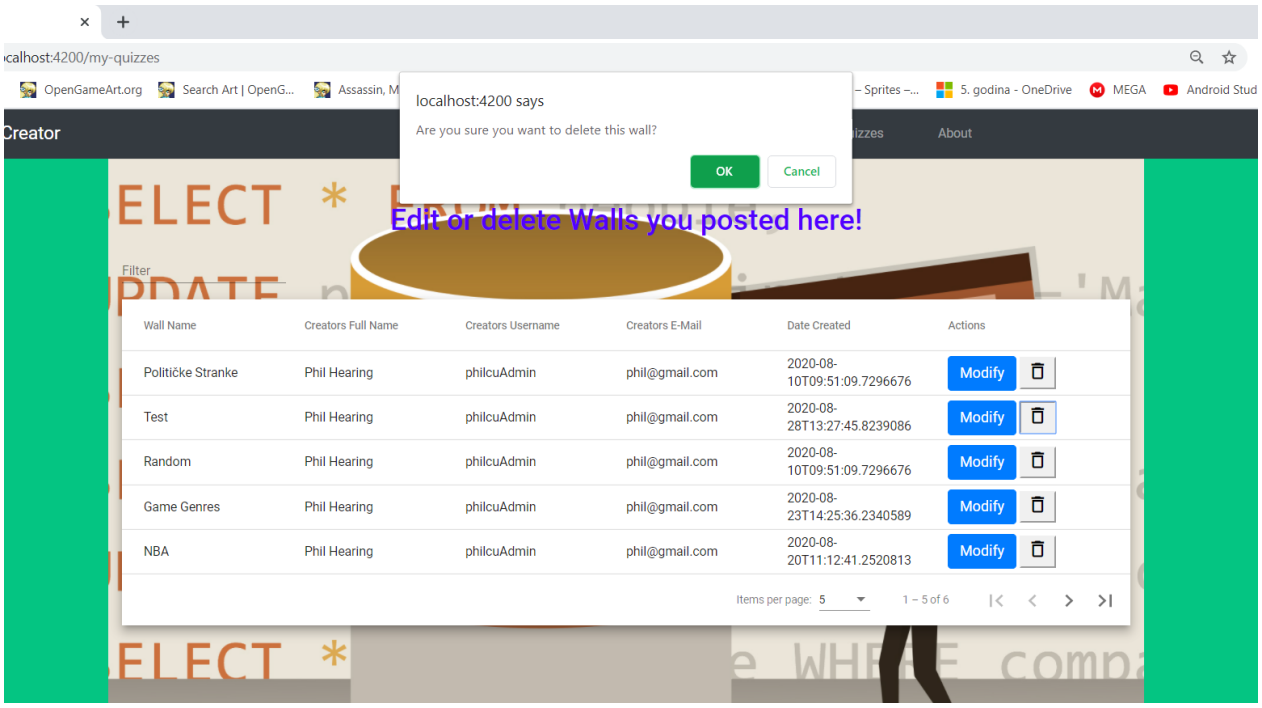
Slika 4.44. *Funkcija DeleteWall koja se izvodi na klik gumba. Prije brisanja pojavljuje se skočni prozor za potvrdu akcije brisanja.*



Slika 4.45. *Prikaz Angular Material podatkovne tablice u komponenti MyQuizzes.*



Slika 4.46. *Prikaz svih registriranih korisnika i svih napravljenih zidova aplikacije u Angular Material tablicama u Admin Panel komponenti.*



Slika 4.47. *Prozor kojim se potvrđuje brisanje odabranog zida.*

4.3.6 Otvaranje skočnih prozora, ažuriranje zida

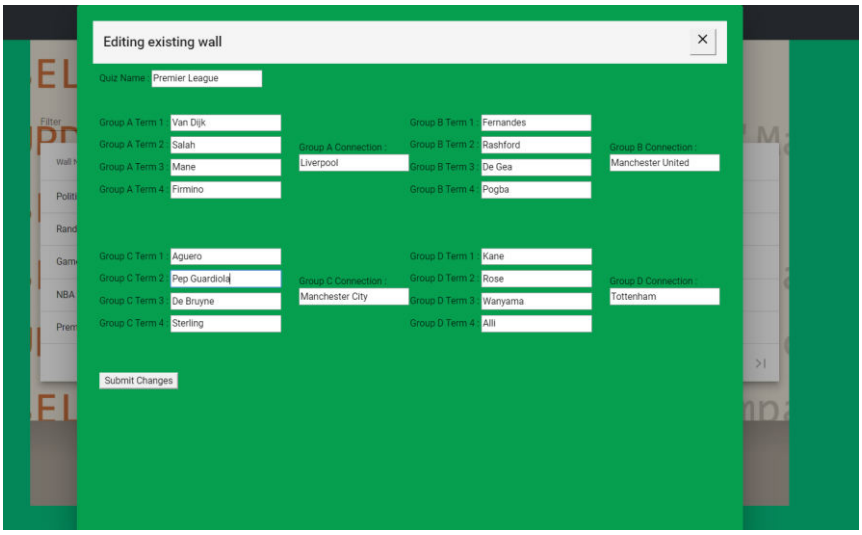
Korisnicima aplikacije potrebno je omogućiti igranje kviza koji se odabere u tablici na Home komponenti, te ažuriranje odabranog zida na MyQuizzes komponenti. Stvaranje skočnih prozora omogućilo je korištenje Angular MatDialogConfig biblioteke. Pomoću te biblioteke instancira se objekt klase MatDialogConfig u kojemu se zadaju svojstva skočnog prozora koji će se otvoriti. Svojstva koja se moraju odrediti su : onemogućavanje zatvaranja (*eng. disable close*), prikaz na sredini stranice (autoFocus), te širina i visina skočnog prozora. Kod otvaranja prozora funkcijom open određuje se komponenta koja će biti otvorena u njemu – klikom na „Modify“ gumb komponente MyQuizzes otvara se komponenta EditWall, a klikom na „Play Quiz“ gumb komponenta PlayQuiz. Kod komponente EditWall postoji forma analogna onoj u komponenti CreateQuiz, a ona prvo dohvaća podatke s poslužitelja i upisuje ih u formu. Ti dohvaćeni podaci mogu se izmijeniti klikom na „Submit Changes“ gumb, čime se radi HTTP Put zahtjev prema poslužitelju.

```
editWall(wallID:string)//editing created walls!
{
  this.WallDetails=this.service.getWallById(wallID).subscribe(
    res=>{
      this.WallDetails=res;
      this.service.populateForm(this.WallDetails);
    }
  );
  const dialogConfig=new MatDialogConfig();
  dialogConfig.disableClose=true;
  dialogConfig.autoFocus=true;
  dialogConfig.width="60%";
  dialogConfig.width="1000px";
  dialogConfig.height="800px";
  this.dialog.open(EditWallComponent,dialogConfig);
}
```

Slika 4.48. *Funkcija za otvaranje skočnog prozora unutar MyQuizzes TypeScript komponente. Funkcija iz servisa populateForm služi za popunjavanje forme podacima odgovarajućeg zida.*

```
1.  populateForm(Object:Wall)
2.    {
3.      this.formModel.patchValue({
4.        wallID:Object.wallID,
5.        wallName:Object.wallName,
6.        GroupATerms:{
7.          GroupATerm1:Object.groupATerms[0].termName,
8.          GroupATerm2:Object.groupATerms[1].termName,
9.          GroupATerm3:Object.groupATerms[2].termName,
10.         GroupATerm4:Object.groupATerms[3].termName
11.       },
12.       GroupBTerms:{
13.         GroupBTerm1:Object.groupBTerms[0].termName,
14.         GroupBTerm2:Object.groupBTerms[1].termName,
15.         GroupBTerm3:Object.groupBTerms[2].termName,
16.         GroupBTerm4:Object.groupBTerms[3].termName
17.       },
18.       GroupCTerms:{
19.         GroupCTerm1:Object.groupCTerms[0].termName,
20.         GroupCTerm2:Object.groupCTerms[1].termName,
21.         GroupCTerm3:Object.groupCTerms[2].termName,
22.         GroupCTerm4:Object.groupCTerms[3].termName
23.       },
24.       GroupDTerms:{
25.         GroupDTerm1:Object.groupDTerms[0].termName,
26.         GroupDTerm2:Object.groupDTerms[1].termName,
27.         GroupDTerm3:Object.groupDTerms[2].termName,
28.         GroupDTerm4:Object.groupDTerms[3].termName
29.       },
30.       GroupAConnections:Object.groupAConnections[0].connectionName,
31.       GroupBConnections:Object.groupBConnections[0].connectionName,
32.       GroupCConnections:Object.groupCConnections[0].connectionName,
33.       GroupDConnections:Object.groupDConnections[0].connectionName,
34.     })
35.   }
```

Slika 4.49. *Funkcija populateForm koja prima objekt zid i popunjava formu sa svojstvima predanog objekta.*



Slika 4.50. *Skočni prozor koji sadrži formu za izmjenu postojećeg zida.*

4.3.7 Implementacija igranja kviza

Unutar Home komponente omogućeno je otvaranje određenog kviza (zida za spajanje) klikom na gumb PlayQuiz. Način otvaranja skočnog prozora ekvivalentan je otvaranju skočnog prozora na MyQuizzes komponenti. Na klik gumba PlayQuiz poziva se funkcija openPlayWindow koja prima identifikator zida iz tablice, dohvaća određeni zid po identifikatoru, otvara skočni prozor s određenim svojstvima, te šalje podatke o zidu komponenti koja će biti otvorena u skočnom prozoru. Komponenta koja je otvorena u skočnom prozoru je PlayQuiz komponenta kojoj je identifikator zida koji se igra poslan iz roditeljske komponente, a preko naredbe Inject je učitan u samu komponentu. Nakon što je podatak identifikator učitan, radi se zahtjev dohvaćanja zida po identifikatoru, te se u HTML dijelu prikazuje ime zida i ime korisnika koji je napravio zid. Pritiskom na gumb Start komponenta PlayQuiz se sakriva, a komponenta Game se pokreće.

```
openPlayWindow(wallID:string)
{
  this.response=this.wallService.getWallById(wallID).subscribe(
    res=>{
      this.WallDetails=res;
      //console.log(this.WallDetails);
    }
  );
  const dialogConfig=new MatDialogConfig();
  dialogConfig.autoFocus=true;
  dialogConfig.disableClose=true;
  dialogConfig.width="60%";
  dialogConfig.width="1000px";
  dialogConfig.height="800px";
  dialogConfig.data=
  {
    wallID:this.WallDetails.wallID
  }
  this.dialog.open(QuizStartComponent,dialogConfig);
}
}
```

Slika 4.51. *Otvaranje skočnog prozora unutar Home komponente. Slanje identifikatora zida komponenti koja se otvara u skočnom prozoru.*

```
constructor(private service:WallService,private userService:UserService,private toastr:
ToastrService, public dialogRef:MatDialogRef<HomeComponent>, @Inject(MAT_DIALOG_DATA)
 public receivedData:any,private dialog:MatDialog)
 {
   this.wallID=this.receivedData.wallID;
 }
```

Slika 4.52. *Hvatanje podataka poslanog iz Home komponente, te njegovo učitavanje u Game komponentu i PlayQuiz komponentu.*

Unutar Game komponente dohvaća se zid po identifikatoru, te se svih 16 pojmova zida sprema u posebne objekte klase Term, klase koja se sastoji od atributa ime pojma i ime grupe. Ti objekti koji predstavljaju pojmove zida će se spremiti u dva polja : prvo polje u kojem će se pojmovi izmiješati (potrebno za prvi dio kviza gdje se pojmovi moraju grupirati) i drugo polje s grupiranim pojmovima (potrebno za drugi dio kviza gdje se pogađaju veze između grupa pojmova).

```
ngOnInit(): void {
    //#region DataGetting
    this.service.getWallById(this.wallID).subscribe(
      res=>
      {
       this.wallDetails=res;
       this.wallName=this.wallDetails.wallName;
       this.termA1.termName=this.wallDetails.groupATerms[0].termName;
       this.termA1.connectionName=this.wallDetails.groupAConnections[0].connectionName;
       this.termA2.termName=this.wallDetails.groupATerms[1].termName;
       this.termA2.connectionName=this.wallDetails.groupAConnections[0].connectionName;
       this.termA3.termName=this.wallDetails.groupATerms[2].termName;
       this.termA3.connectionName=this.wallDetails.groupAConnections[0].connectionName;
       this.termA4.termName=this.wallDetails.groupATerms[3].termName;
       this.termA4.connectionName=this.wallDetails.groupAConnections[0].connectionName;
       this.termB1.termName=this.wallDetails.groupBTerms[0].termName;
       this.termB1.connectionName=this.wallDetails.groupBConnections[0].connectionName;
       this.termB2.termName=this.wallDetails.groupBTerms[1].termName;
       this.termB2.connectionName=this.wallDetails.groupBConnections[0].connectionName;
       this.termB3.termName=this.wallDetails.groupBTerms[2].termName;
       this.termB3.connectionName=this.wallDetails.groupBConnections[0].connectionName;
       this.termB4.termName=this.wallDetails.groupBTerms[3].termName;
       this.termB4.connectionName=this.wallDetails.groupBConnections[0].connectionName;
       this.termC1.termName=this.wallDetails.groupCTerms[0].termName;
       this.termC1.connectionName=this.wallDetails.groupCConnections[0].connectionName;
       this.termC2.termName=this.wallDetails.groupCTerms[1].termName;
       this.termC2.connectionName=this.wallDetails.groupCConnections[0].connectionName;
       this.termC3.termName=this.wallDetails.groupCTerms[2].termName;
       this.termC3.connectionName=this.wallDetails.groupCConnections[0].connectionName;
       this.termC4.termName=this.wallDetails.groupCTerms[3].termName;
       this.termC4.connectionName=this.wallDetails.groupCConnections[0].connectionName;
       this.termD1.termName=this.wallDetails.groupDTerms[0].termName;
       this.termD1.connectionName=this.wallDetails.groupDConnections[0].connectionName;
       this.termD2.termName=this.wallDetails.groupDTerms[1].termName;
       this.termD2.connectionName=this.wallDetails.groupDConnections[0].connectionName;
       this.termD3.termName=this.wallDetails.groupDTerms[2].termName;
       this.termD3.connectionName=this.wallDetails.groupDConnections[0].connectionName;
       this.termD4.termName=this.wallDetails.groupDTerms[3].termName;
       this.termD4.connectionName=this.wallDetails.groupDConnections[0].connectionName;
        this.terms = [
          this.termA1,this.termA2,this.termA3,this.termA4, //za grupiranje pojmova
          this.termB1,this.termB2,this.termB3,this.termB4,
          this.termC1,this.termC2,this.termC3,this.termC4,
```

```
44.           this.termD1,this.termD2,this.termD3,this.termD4];
45.         this.sortedTerms=[ ,//za pogađanje veza između pojmova
46.           this.termA1,this.termA2,this.termA3,this.termA4
47.           this.termB1,this.termB2,this.termB3,this.termB4,
48.           this.termC1,this.termC2,this.termC3,this.termC4,
49.           this.termD1,this.termD2,this.termD3,this.termD4];
```

Slika 4.53. *Dohvaćanje zida po identifikatoru, spremanje pojmova zida u objekte klase Term, te spremanje objekata u polje za poredane i neporedane pojmove.*

Nakon spremanja podataka u polja, potrebno je izmiješati pojmove u polju terms korištenjem Fisher-Yates algoritma. Fisher-Yatesov algoritam podrazumijeva prolaženje kroz elemente polja od zadnjeg elementa sve dok se ne dođe do prvog elementa polja, a u svakoj iteraciji zamjenjuje trenutni element polja s elementom polja koji se nalazi na nasumičnom indeksu (indeks je manji ili jednak indeksu trenutnog elementa). Ovaj algoritam implementiran je funkcijom Shuffle.

```
1.  Shuffle(Array)//Fisher-Yates shuffle
2.    {
3.     var currentIndex=Array.length;//trenutni indeks je indeks posljednjeg elementa
4.     var tempValue,randomIndex;//tempValue služi za zamjenu dva elementa, randomIndex je
    nasumični indeks s kojim se mijenja trenutni element
5.     while(currentIndex!=0) //ponovi sve dok trenutni indeks nije 0, dok se ne dođe do pr
    vog elementa
6.     {
7.        randomIndex=Math.floor(Math.random()*currentIndex);//generiranje nasumičnog broja
     između 0 i 1, njegova pretvorba u manji cijeli broj
8.        currentIndex--; //pomicanje na element s manjim indeksom
9.        tempValue=Array[currentIndex]; //promjena elementa s nasumičnog indeksa i element
    a na trenutnom indeksu
10.       Array[currentIndex]=Array[randomIndex];
11.       Array[randomIndex]=tempValue;
12.    }
13.    return Array; //vraćanje izmješanog polja
14.   }
```

Slika 4.54. *Funkcija Shuffle koja izvodi miješanje elemenata polja po Fisher-Yatesovom algoritmu.*

Svaka pločica u HTML dijelu dokumenta sadržavat će pojam iz polja terms, a njezin identifikator bit će ime grupe kojoj pojam pripada. Klikom na pločicu poziva se funkcija pushInCheckArray. Ako je pojam pritisnut, grupa kojoj pojam pripada (identifikator pritisnutog pojma) ubacuje se u polje Group, a ako korisnik ponovno pritisne na pojam, grupa pritisnutog pojma izbacuje se iz polja. Kad broj pojmova u polju Group dosegne četiri (maksimalan broj elemenata u grupi), poziva se funkcija checkForTermMatch koja prima polje Group i provjerava jesu li svi elementi unutar polja isti. Ako jesu, rezultat se povećava, a svi pojmovi koji su ispravno izabrani više ne mogu biti pritisnuti i unešeni u polje Group. To je izvedeno funkcijom hideElement koja mijenja CSS klasu pločica kojima je identifikator jednak pojmovima u polju Group. Kad broj uparenih grupa dosegne dva, korisniku se prikazuju tri života, odnosno broj pokušaja da grupira preostale dvije grupe pojmova. Ako svi elementi unutar polja Group nisu

isti, broj pritisnutih pojmova postavlja se u nulu, te se smanjuje broj života ako je broj uparenih grupa veći od 2. Kad broj života dosegne 0, korisnik se preusmjerava na dio gdje može ostvariti dodatne bodove tako da pogodi veze između uparenih grupa pojmova. Korisnik se na taj dio preusmjerava i kad ispravno upari sve pojmove kao i kad ih ne uspije upariti u vremenu od tri minute.

```
1.  <div class="tile" style="left:40px;top:55px;" (click)="isClickedtile1=!isClickedtile1;
    chosenSound.play();PushInCheckArray($event,isClickedtile1);getParagraphText($event);" [
    class.ClickedTile]="isClickedtile1" id="{{this.terms[0].connectionName}}">
2.          <p hidden>{{this.terms[0].connectionName}}</p>
3.          <p>{{this.terms[0].termName}}</p>
4.      </div>
```

Slika 4.55. *Prikaz pločice koja sadrži pojam zida. Kad je pritisnuta, stanje pritisnuta/nije pritisnuta se mijenja.*

```
1.  PushInCheckArray(event,isClicked)
2.    {
3.      var con=this.getTermConnection(event);  //dohvaćanje grupe kojoj pojam pripada
4.      if(isClicked==true)  //Ako je pojam pritisnut
5.      {
6.      if(con=="")   //upozorenje ako se grupa ne uspije dohvatiti
7.      {
8.        this.toastr.warning("Please Unselect and Select the term again","Could not get ID
     of clicked Term!");
9.      }
10.     this.numOfClickedTerms++;  //Broj pritisnutih pojmova se povećava
11.     this.Group.push(con);    //stavljanje grupe u polje
12.       if(this.Group.length==4)  //Ako je duljina polja Group jednaka 4
13.       {
14.         this.checkForTermMatch(this.Group);  //pozivi funkciju checkForTermMatch
15.         this.Group=[];      //isprazni polje Group      }
16.     }
17.     else if(isClicked==false)  //Ako pojam nije pritisnut
18.     {
19.       this.numOfClickedTerms--;  //smanji broj pritisnutih pojmova
20.       this.Group.splice(this.numOfClickedTerms,1); //izbaci element iz polja
21.     }
22.   }
```

Slika 4.56. *Funkcija koja se poziva klikom na pločicu koja sadrži pojam zida.*

```
1.  checkForTermMatch(array)
2.    {
3.     const allEqual=array=>array.every(v=>v===array[0]);//funkcija koja vraća istinu ako
    su svi elementi predanog polja jednaki
4.      if(array.length==4 && allEqual(array)==true) //Ako je duljina polja jednaka 4 i ako
    su svi elementi polja jednaki
5.      {
6.        var Connection=array[0];  //uzmi ime grupe
7.        console.log("Konekcija :"+Connection);
8.        console.log("Matched properly!");
9.        this.score++; //povećaj rezultat
10.       this.numOfPairedGroupTerms++; //povećaj broj grupiranih pojmova
11.       console.log(this.numOfPairedGroupTerms);
```

```
12.       this.numOfClickedTerms=0;  //postavi da niti jedan pojam nije odabran
13.       this.successSound.play();  //zvuk uspješnog grupiranja
14.       this.hideElement(Connection);  //promjena klase HTML elementa uspješno povezane
    grupe pojmova
15.       this.AllIsUnclicked();  //svi pojmovi više nisu odabrani
16.       if(this.numOfPairedGroupTerms==2)  //ako je broj grupiranih pojmova jednak 2
17.       {
18.         this.showLives=true; //pokazi zivote
19.       }
20.
21.       if(this.numOfPairedGroupTerms==3)  //Ako je broj grupiranih pojmova jednak 3
22.       {
23.         this.score=4;  //rezultat je 4, sve je ispravno grupirano
24.         this.gameOver=true; //idi na pogađanje konekcija
25.         this.successfullyGroupedTheTerms=true;  //prikaz poruke ako sus vi pojmovi
    uspješno vezani
26.       }
27.     }
28.     else if(array.length==4 && allEqual(array)==false)  //ako su krivo odabrana 4 pojma
29.     {
30.       console.log("Not matched properly!");
31.       this.wrongSound.play();  //zvuk krivog odabira
32.       this.AllIsUnclicked();  //svi pojmovi nisu više odabrani
33.       this.numOfClickedTerms=0;  //broj pritisnutih pojmova je 0
34.       if(this.numOfPairedGroupTerms>=2)  //ako je korisnik grupirao više od 2 grupe
    pojmova
35.       {
36.         this.numOfLives--; //smanji broj života
37.         console.log("Lives:"+this.numOfLives);
38.         if(this.numOfLives==2)  //ako je broj života jednak 2
39.         {
40.           this.firstLifeWasted=true;  //prvi život je potrošen
41.         }
42.         else if(this.numOfLives==1)  //ako je broj života jednak 1
43.         {
44.           this.secondLifeWasted=true;  //drugi život je potrošen
45.         }
46.         else if(this.numOfLives==0)  //Ako je broj života jednak 0
47.         {
48.           this.thirdLifeWasted=true;  //treći život je potrošen
49.           console.log("Game Over");//ovdje funkcija za game over
50.           this.gameOver=true;  //otvaranje prozora Game Over
51.           this.wastedLives=true;  //poruka ako je korisnik potrošio sve živote
52.         }
53.       }
54.     }
55.   }
```

Slika 4.57. *Funkcija za provjeru ispravnog grupiranja pojmova.*

Za implementaciju odbrojavanja vremena rješavanja kviza koristila se biblioteka CountDownModule koja je u HTML dokumentu omogućila countdown element koji prikazuje odbrojavanje. Kad odbrojavanje dosegne nulu, poziva se funkcija timesUp koja igrača vodi na Game Over dio.

```
<div class="Time" *ngIf="gameOver==false">
    <label>Time left : </label>
<p><strong><countdown [config]="{leftTime:180,format:'m:s'}"(event)="timesUp($event)">
</countdown></strong></p>
</div>
```

Slika 4.58. *Prikaz HTML oznake za odbrojavanje vremena rješavanja kviza u Game komponenti.*

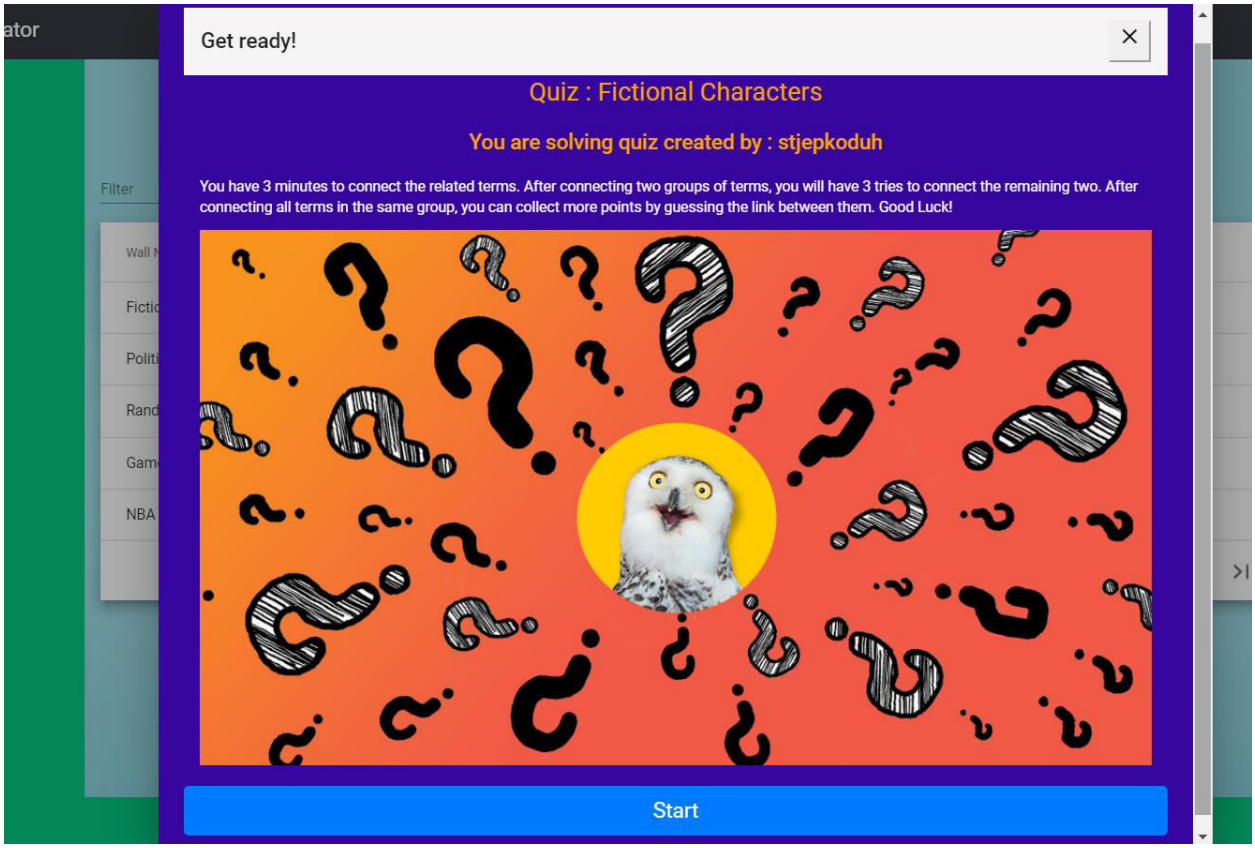
```
timesUp(event)
  {
    if(event.action=="done")
    {
      console.log("Time expired!");
      this.gameOver=true;
      this.ranOutOfTime=true;
    }
  }
```

Slika 4.59. *Funkcija timesUp koja se poziva kad vrijeme odbrojavanja istekne. Postavljanje vrijednosti varijabli gameOver i ranOutOfTime o kojima ovisi prikaz HTML elemenata komponente.*

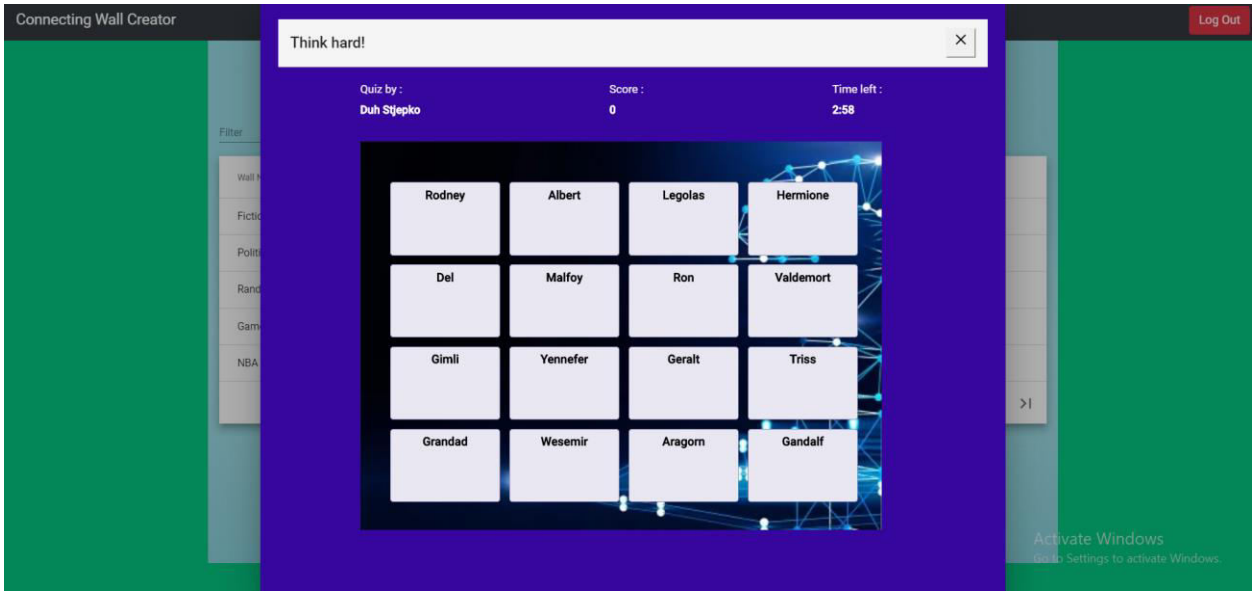
Prilikom stvaranja kviza dodani su i zvučni efekti koji se aktiviraju pri odabiru pločice pojma, pri uspješnom grupiranju pojmova (uspješno pogođenoj vezi) i pri neuspješnom grupiranju pojmova. Manipulacija zvukom i definiranje varijabli koje spremaju zvučne zapise omogućeno je korištenjem Howler JavaScript modula.

```
import {Howl} from 'howler';  //unos Howler modula u komponentu
...
successSound=new Howl({  //zvuk uspješnog grupiranja
    src:['../../assets/soundEffects/success.mp3'],
    volume: 1
  });
  chosenSound=new Howl({  //zvuk odabranog pojma
    src:['../../assets/soundEffects/match.wav'],
    volume: 0.2
  });
  wrongSound=new Howl({  //zvuk neuspješnog grupiranja
    src:['../../assets/soundEffects/wrong.mp3'],
    volume: 0.5
  });
```

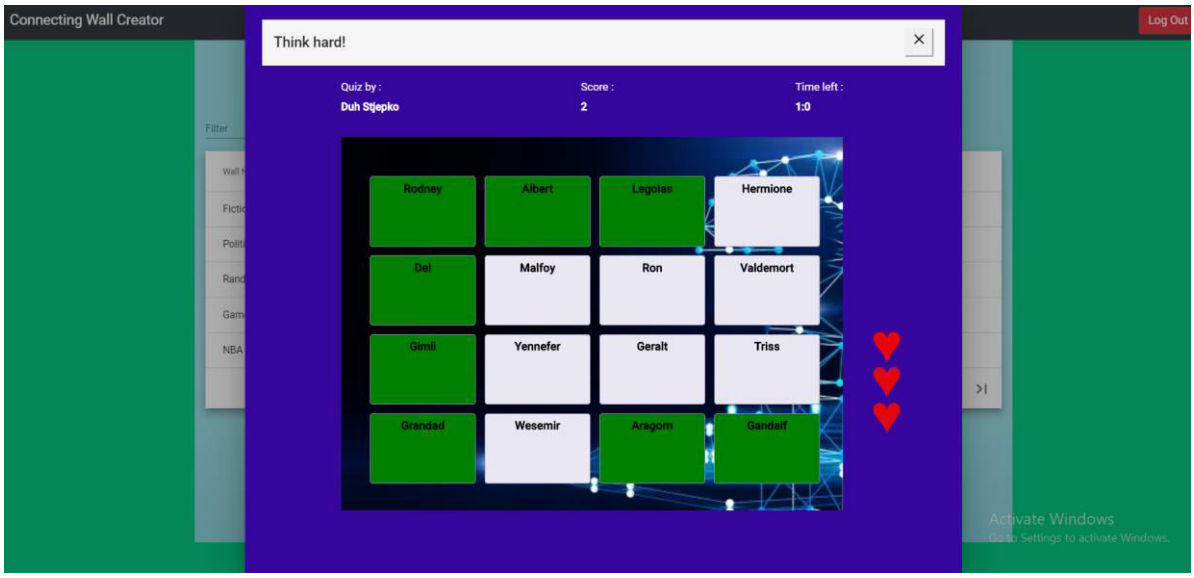
Slika 4.60. *Varijable u koje se unose zvučni zapisi.*



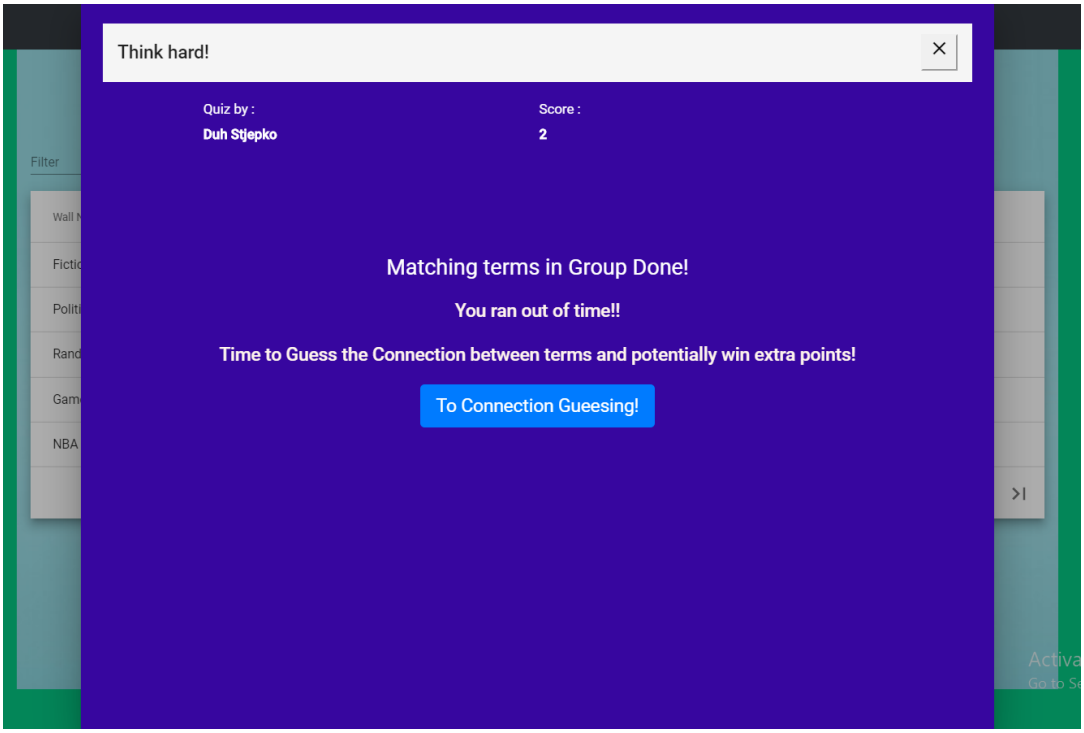
Slika 4.61. *Prikaz skočnog prozora otvorenog na Home komponenti.*



Slika 4.62. *Igranje kviza „Zid za spajanje“. Dio kviza gdje se grupiraju pojmovi.*



Slika 4.63. *Pojavljivanje broja života kad igrač uspije grupirati dvije grupe pojmova.*



Slika 4.64. *Zaslon koji se prikazuje ako korisnik uspješno grupira sve pojmove, ako potroši živote ili ako ne uspije grupirati pojmove u tri minute.*

Nakon završetka uparivanja pojmova, igrač se vodi na dio na kojem se prikazuje poruka koja mu govori je li uspješno grupirao pojmove, je li mu vrijeme isteklo ili je li potrošio sva tri života (Slika 4.3.7.5). Klikom na gumb „To Connection Guessing!“ igrač se vodi na dio igre gdje

pogađa veze između grupiranih pojmova. Na dijelu gdje igrač pogađa veze pored navedenih pojmova svake grupe prisutan je unos teksta gdje se treba navesti po čemu su pojmovi tog reda povezani. Igrač potvrđuje unesenu vezu pritiskom na gumb „Confirm“ ispod unosa za tekst. Ako igrač pogodi vezu, umjesto gumba se pojavljuje poruka o točno pogođenoj vezi, a ako unese netočnu vezu poruka da nije točno, te ispravna veza između pojmova (točan odgovor). Rezultat koji je korisnik postigao povećava se pri svakom ispravnom određivanju veze. Dakle, maksimalan broj bodova koji korisnik može ostvariti na kvizu je osam. Valja napomenuti da odbrojavanje prestaje kad dio igre s grupiranjem pojmova završi, te da korisnik ima neograničeno vrijeme za pogoditi veze između pojmova.

Slika 4.65. *Dio kviza gdje igrač pogađa vezu između grupa pojmova.*

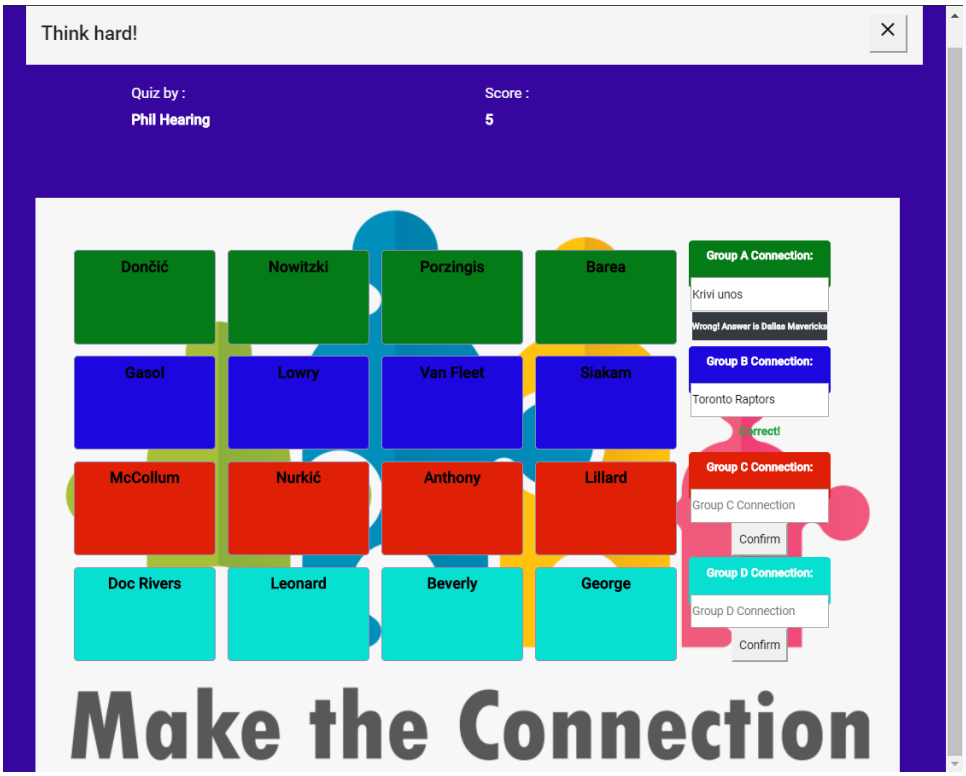
Kad korisnik unese vezu u tekstualno polje, ta veza se sprema u četiri varijable guessedConnection gdje svaka od varijabli predstavlja vezu između pojmova grupa. Funkcijama getConnection dohvaća se uneseni tekst, sprema se u varijable, te se uspoređuje s ispravnom vezom između pojmova. Ako je veza točna, rezultat se povećava. Kad igrač pogodi (ispravno ili neispravno) veze za sve grupe pojmova, vodi ga se na prikaz njegovog konačnog rezultata s prikladnom porukom.

```
getConnection1()
{
  var conn=(<HTMLInputElement>document.getElementById("conn1")).value; //dohvaćanje teksta iz tekstualnog unosa po identifikatoru
  this.guessedConnection1=conn; //spremanje teksta u varijablu
  if(this.guessedConnection1==this.termA1.connectionName) //uspoređivanje unesene veze i ispravne veze
  {
    this.score++; //povećaj rezultat
    this.successSound.play();//zvuk ispravnog pogađanja
  }
  else
  {
    this.wrongSound.play(); //zvuk neispravnog pogađanja
  }
  this.firstConnectionGuessed=true; //varijabla koja kazuje da je prva veza pogođena
  console.log(this.guessedConnection1);
}
```

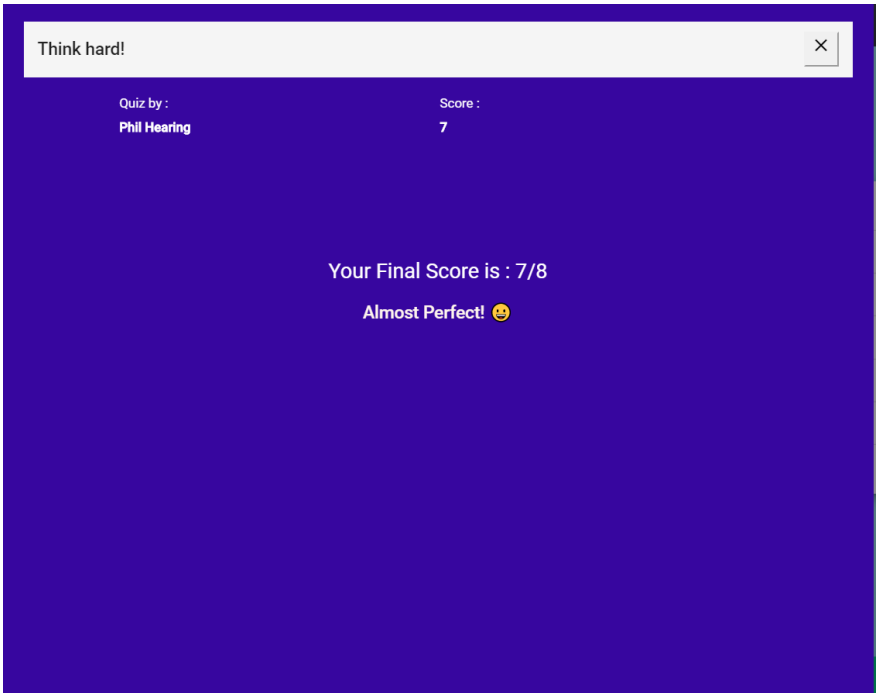
Slika 4.66. *Prikaz dohvaćanja pogođene veze iz tekstualnog unosa i provjera je li unešena veza točna. Ekvivalentne funkcije napisane su za pogađanje preostale tri veze.*

```
checkIfUserGuessedAllConnections() //provjera jesu li sve veze pogođene
{
  if(this.guessedConnection1!=null && this.guessedConnection2!=null && this.guessedConnection3!=null && this.guessedConnection4!=null)
  {
    return true;
  }
  else
  {
    return false;
  }
}
```

Slika 4.67. *Funkcija koja provjerava je li korisnik pogodio (ispravno ili neispravno) sve veze između pojmova. Krajnji rezultat kviza prikazuje se ako funkcija vraća logičku istinu.*



Slika 4.68. *Prikaz netočnog unosa veze za prvu grupu pojmova, te točnog unosa za drugu grupu pojmova.*



Slika 4.69. *Prikaz rezultata kviza koji se prikazuje kad korisnik pogodi sve veze između grupa pojmova.*

5. ZAKLJUČAK

Web aplikacija razvijena za ovaj diplomski rad je online kviz „Zid za spajanje“. Za pristup aplikaciji potrebna je registracija i prijava korisnika, pri čemu aplikacija jamči zadovoljavajuću razinu zaštite korisničkih podataka. Uz rješavanje kvizova koje su napravili drugi korisnici, napravljena aplikacija korisnicima omogućuje kreiranje vlastitih kvizova, ažuriranje napravljenih kvizova, te njihovo brisanje. Napravljena aplikacija može se koristiti u Pub kvizovima ili za provjeru znanja djece u predškolskoj dobi ili nižim razredima. Ovakav tip kviza stekao je popularnost u Ujedinjenom Kraljevstvu zbog TV kviza „Connecting Wall“, a ova aplikacija omogućuje korisnicima igranje ovakve vrste kviza online, te objavu napravljenih kvizova za svoje prijatelje, učenike i sl. Za razliku od ostalih online aplikacija iste svrhe (PuzzGrid i ClassTools), aplikacija ima i funkcionalnost pretrage zidova po imenu zida, kao i mogućnost ažuriranja postojećeg zida. Aplikacija ponekad ima greške s performansama kod igranja kviza, točnije kod biranja pojma prilikom njihovog grupiranja, no igrač je upozoren kako se rješava takav problem. Kod izrade aplikacije korištene su brojne tehnologije i programerski okviri u kojima se razvija velik broj komercijalnih poslovnih aplikacija. Aplikacija je na engleskom jeziku, a jednostavna je i intuitivna za korištenje ljudima koji pripadaju i drugim govornim područjima.

LITERATURA

[1] A.F, Adam Freeman, Pro ASP.NET Core MVC, izdavač: AlITeBooks, šesto izdanje, 2016. (posljednji pristup : Kolovoz 2020.)

[2] A.F, Adam Freeman, Essential Anglar for ASP.NET Core MVC, izdavač : AllITeBooks, prvo izdanje, 2017. (posljednji pristup : Kolovoz 2020.)

[3] W3Schools, SQL jezik : https://www.w3schools.com/sql/ (posljednji pristup : Lipanj 2020.)

[4] Microsoft, ASP.NET Core : https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-2.2 (posljednji pristup : Lipanj 2020.)

[5] Microsoft, MVC u ASP.NET-u : https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-3.1 (posljednji pristup : Lipanj 2020.)

[6] Lucidchart, Microsoft Visio alat : https://www.lucidchart.com/pages/what-is-microsoft-visio#section_0 (posljednji pristup : Lipanj 2020.)

[7] EntityFrameworkTutorial, Entity Framework : https://www.entityframeworktutorial.net/what-is-entityframework.aspx (posljednji pristup : Lipanj 2020.)

[8] W3Schools, C# programski jezik : https://www.w3schools.com/cs/ (posljednji pristup : Lipanj 2020.)

[9] Microsoft, ASP.NET Web API : https://dotnet.microsoft.com/apps/aspnet/apis (posljednji pristup : Lipanj 2020.)

[10] Postman : https://www.postman.com/about-postman/ (posljednji pristup : Lipanj 2020.)

[11] Angular : https://angular.io/docs (posljednji pristup : Lipanj 2020.)

[12] HTML jezik : https://www.w3schools.com/html/ (posljednji pristup : Lipanj 2020.)

[13] CSS jezik : https://www.w3schools.com/css/ (posljednji pristup : Lipanj 2020.)

[14] TypeScript jezik : https://www.typescriptlang.org/ (posljednji pristup : Lipanj 2020.)

[15] NGX Bootstrap : https://www.npmjs.com/package/ngx-bootstrap (posljednji pristup : Lipanj 2020.)

[16] Visual Studio Code : https://code.visualstudio.com/ (posljednji pristup : Lipanj 2020.)

[17] Zid za spajanje, slika : https://tenminutesforme.wordpress.com/2012/07/09/train-your-mind-play-the-only-connect-wall/ (posljednji pristup : Kolovoz 2020.)

[18] MVC arhitektura, slika : https://blog.cloudboost.io/what-is-model-view-controller-124a9942246

[19] Slika Entity Framework arhitekture : https://www.codeproject.com/Articles/363040/An-Introduction-to-Entity-Framework-for-Absolute-B

[20] Code-First konvencije u Entity Frameworku : https://www.entityframeworktutorial.net/code-first/configure-one-to-many-relationship-in-code-first.aspx

[21] Demonstracija enkripcije JWT tokenom : https://jwt.io/#libraries-io

[22] Opis JWT Tokena : https://jwt.io/introduction/ (posljednji pristup : Kolovoz 2020.)

[23] Konvencije stvaranja modela u Entity Framework okruženju : https://www.entityframeworktutorial.net/efcore/one-to-many-conventions-entity-framework-core.aspx (posljednji pristup : Lipanj 2020.)

[24] Kaskadno brisanje u Entity Framework okruženju : https://docs.microsoft.com/en-us/ef/core/saving/cascade-delete (posljednji pristup : Lipanj 2020.)

[25] Fisher-Yates algoritam miješanja : https://bost.ocks.org/mike/shuffle/

[26] Omogućavanje dijeljenja resursa s drugom aplikacijom (CORS) : https://docs.microsoft.com/en-us/aspnet/core/security/cors?view=aspnetcore-3.1 (posljednji pristup : Kolovoz 2020.)

[27] Angular 2+ forme i validacija : https://angular-templates.io/tutorials/about/angular-forms-and-validations (posljednji pristup : Kolovoz 2020.)

SAŽETAK

Diplomski rad opisuje izradu web aplikacije za igranje online kviza. Rad sadrži opis tijeka razvoja programske podrške od projektiranja relacijske baze podataka u SQL-u, preko izrade poslužiteljske aplikacije u .NET okviru, do izrade korisničkog sučelja u Angular 9 okviru. Aplikacija unutar Entity Framework okvira razvijena je po Model-Pogled-Upravljač obrascu razvoja programske podrške. Prikazane su mogućnosti i arhitektura razvojnog .NET Entity Framework razvojnog okvira u kojem se radio poslužiteljski dio aplikacije, kao i značajke Angular 9 okvira u kojemu se projektiralo korisničko sučelje. Na korisničkom sučelju omogućeno je igranje kvizova, kao i dohvaćanje napravljenih kvizova i njihov jasan prikaz u tablici. Svrha napravljene aplikacije „Zid za spajanje“ je omogućiti korisniku igranje kvizova koje su napravili drugi korisnici, te kreiranje i objavu vlastitog kviza. Korisniku je također omogućeno brisanje kviza i njegovo ažuriranje. Aplikacija omogućuje zanimljivo iskustvo igranja kviza u kojemu je potrebno grupirati šesnaest pojmova u četiri grupe po skrivenoj vezi između pojmova, te pogoditi po čemu su pojmovi povezani.

Ključne riječi : Angular 9, ASP.NET Entity Framework, C#, Code First, MVC, Online kviz, SQL

ABSTRACT

AN INTERACTIVE ONLINE QUIZ

The Graduate paper describes the development of a web application for playing Online Quiz. The Paper consists of describing the process of software development which consists of designing an SQL relational database, creating a Web API server application (Backend) in ASP.NET and creating a user interface in web application (Frontend) with Angular 9 framework. The development of application in Entity Framework was done by following MVC (Model-View-Controller) software design pattern. The paper describes the features and architecture of .NET Entity Framework which was used for Web API creation, as well as features of Angular 9 Framework which was used for the design and creation of user interface. The user interface allows Quiz playing as well as getting all the created Quizzes from database and displaying them in tables. The purpose of the application „Connecting Wall Creator“ is allowing users to play Quizzes created by others as well as creating and sharing their own Quizzes. Users are enabled to delete and update their own Quizzes. Using application provides a thrilling experience of playing Quiz where the goal is to group sixteen terms in four groups by their hidden connection and guessing the hidden connections between them.

Key words : Angular 9, ASP.NET Entity Framework, C#, Code First, MVC, Online Quiz, SQL

ŽIVOTOPIS

Filip Ćuić rođen je 17. svibnja 1996. godine u Našicama. Od rođenja živi u Našicama, gdje stječe osnovnoškolsko obrazovanje. Godine 2011. upisuje Srednju Školu Izidora Kršnjavoga Našice, smjer Opća Gimnazija. Sve razrede od početka do kraja obaveznog školovanja prolazi s odličnim uspjehom. Školske godine 2014./15. završava srednju školu, te akademske godine 2015./16. upisuje prvu godinu preddiplomskog studija računarstva na Elektrotehničkom Fakultetu Osijek koji godinu kasnije mijenja ime u Fakultet elektrotehnike, računarstva i informacijskih tehnologija. Nakon završetka preddiplomskog studija akademske 2017./18. godine upisuje diplomski studij računarstva, smjer Informacijske i podatkovne znanosti. Navedeni studij još uvijek pohađa.

Potpis:

_____

PRILOZI

Prilog 1 : Datoteke pisanog dijela rada (docx i pdf).

Prilog 2 : Programski kod aplikacije Connecting Wall Creator.