Web aplikacija za organizaciju sastanaka i događaja

Marić, Filip

VELEUČILIŠTE U BJELOVARU
STRUČNI PRIJEDIPLOMSKI STUDIJ RAČUNARSTVO

Web aplikacija za organizaciju sastanaka i događaja

Završni rad br. 03/RAČ/2024

Filip Marić

Bjelovar, listopad 2024.

obrazac ZR - 001

Veleučilište u Bjelovaru

Trg E. Kvaternika 4, Bjelovar

1. DEFINIRANJE TEME ZAVRŠNOG RADA I POVJERENSTVA

Student: **Filip Marić** JMBAG: **0314020904**

Naslov rada (tema): **Web aplikacija za organizaciju sastanaka i događaja**

Područje: **Tehničke znanosti** Polje: **Računarstvo**

Grana: **Primijenjeno računarstvo**

Mentor: **Krunoslav Husak, dipl. ing. rač.** zvanje: **viši predavač**

Članovi Povjerenstva za ocjenjivanje i obranu završnog rada:

1. **Tomislav Adamović, mag. ing. el., predsjednik**
2. **Krunoslav Husak, dipl. ing. rač., mentor**
3. **Ivan Sekovanić, mag. ing. inf. et comm. techn., član**

2. ZADATAK ZAVRŠNOG RADA BROJ: 03/RAČ/2024

U sklopu završnog rada potrebno je:

1. Analizirati i opisati .NET razvojni okvir na strani poslužitelja i usporediti s drugim razvojnim okvirima.
2. Analizirati i opisati Blazor razvojni okvir na strani klijenta i usporediti s drugim razvojnim okvirima.
3. Predložiti i opisati relacijski model baze podataka koji će omogućiti pohranu podataka o svim događajima i sastancima.
4. Izraditi i opisati programsko rješenje aplikacije za organizaciju sastanaka na strani poslužitelja koristeći .NET razvojni okvir.
5. Izraditi i opisati programsko rješenje aplikacije za organizaciju sastanaka na strani klijenta koristeći Blazor razvojni okvir.

Datum: 8. svibnja 2024. godine Mentor: **Krunoslav Husak, dipl. ing. rač.**

Zahvala

Zahvaljujem se svom mentoru Krunoslavu Husaku, dipl. ing. rač. na pomoći i savjetovanju tijekom izrade završnog rada kao i tijekom cijelog studija. Također bih se zahvalio ostalim djelatnicima Veleučilišta u Bjelovaru koji su svojim angažmanom i trudom pomogli da steknem nova znanja i dođem do svoga cilja.

Sadržaj

1. Uvod **1**

2. Razvojni okviri na strani poslužitelja **2**

2.1 Django *2*

2.2 Ruby on Rails *3*

2.3 Spring *4*

2.4 .NET *6*

3. Razvojni okviri na strani klijenta **8**

3.1 Angular *8*

3.2 React *9*

3.3 jQuery *10*

3.4 Blazor *11*

4. Relacijski model baze podataka **13**

4.1 Strukturiranje modela baze *13*

4.2 Normalizacija baza podataka *14*

4.3 Sigurnost baza podataka *16*

5. Razvoj web aplikacije **18**

6. ZAKLJUČAK **28**

7. LITERATURA **29**

8. OZNAKE I KRATICE **30**

9. SAŽETAK **31**

10. ABSTRACT **32**

1. Uvod

Svrha ovog rada je upoznati se s .NET tehnologijom u razvoju web aplikacija te prednostima i nedostatcima te tehnologije. Kako bi se spoznale neke mane i prednosti .NET tehnologije u svrhu rada razvijena je web aplikacija za dogovaranje sastanaka. Aplikacija je razvijana unutar dva .NET razvojna okvira, a to su Blazor koji je korišten za razvoj aplikacije na strani klijenta i .NET API koji je korišten za razvoj poslužiteljske strane aplikacije.

U drugom poglavlju uspoređuju se razvojni okviri na strani poslužitelja. Navedena su četiri popularna razvojna okvira te se o svakome mogu steći određena znanja o arhitekturi, primjeni te prednostima i nedostacima u korištenju tih razvojnih okvira

Unutar trećeg poglavlja uspoređuju se razvojni okviri na strani korisnika. Kao i u prošlom poglavlju unutar ovog spominju se četiri razvojna okvira. Svaki razvojni okvir koji se spominje govori o tome koji je cilj primjene, kako funkcionira te za što se koristi, kao i prednostima i nedostatcima rada u njemu.

U četvrtom poglavlju spominje se korištenje baze podataka i zbog čega je bitno imati dobru organizaciju baze podataka. Unutar ovog poglavlja mogu se steći znanja o samom stvaranju baze na početku nekoga projekta, koliko je bitno dobro strukturirati bazu te paziti na integritet podataka i sigurnosne aspekte baze.

Peto poglavlje govori o razvoju aplikacije i koji je njen cilj. U razvoju aplikacije navedene su neke prednosti i nedostaci samog razvoja u razvojnim okvirima Blazor i .NET API te kako se došlo do rješenja da bi aplikacija mogla postići svoju svrhu.

2. Razvojni okviri na strani poslužitelja

Razvojni okviri na strani poslužitelja ključan su element u razvoju modernih web aplikacija. Koristimo ih kako bi spriječili ponavljanje osnovnih postavki i konfiguracija. Na taj način programerima pružaju strukturu i funkcionalnosti koje omogućuju fokus na logiku i specifične zadatke aplikacije.

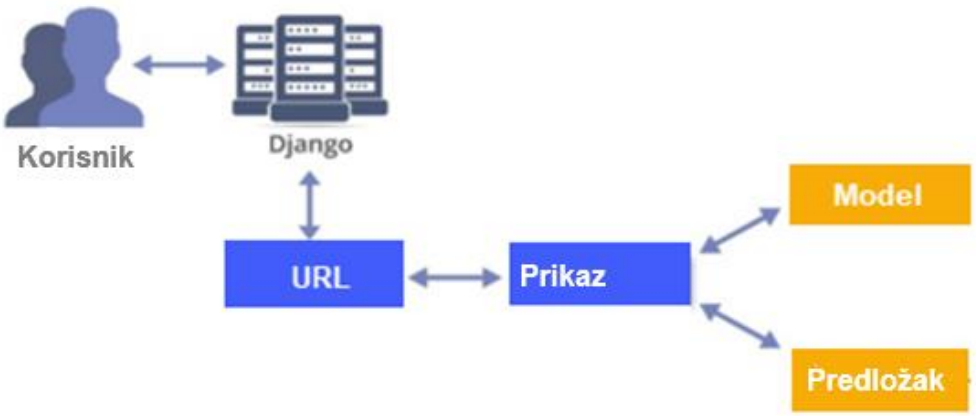
Popularni razvojni okviri na strani poslužitelja su Django, Ruby on Rails, Spring i ASP.NET Core.

Glavni razlog za uporabu razvojnih okvira na poslužiteljskoj strani je sigurnost. Unutar razvojnih okvira su ugrađene sigurnosne značajke koje štite aplikaciju od prijetnji kao što su SQL injekcije, napad umetanjem koda (engl. *Cross-Site Scripting - XSS*) i krivotvorenje među mrežnih zahtjeva (engl. *Cross-Site Request Forgery - CSRF*). Osim sigurnosti, značajno je poboljšana produktivnost korištenjem ovih razvojnih okvira. U suštini, razvojni okviri na strani poslužitelja pružaju strukturu, sigurnost i efikasnost.

2.1 Django

Django je među popularnijim razvojnim okruženjima na strani poslužitelja i nudi sveobuhvatna rješenja za razvoj web aplikacija koristeći Python. Dizajniran je 2005. godine s ciljem da olakša razvoj web aplikacija vođenih bazama podataka. „Primarni cilj ovog web okvira visoke razine je stvoriti complex web stranice koje se temelje na bazi podataka.“ [1]

Arhitektura Djanga temeljena je na MVC obrascu i MVT obrascu. MVC obrazac temelji se na modelu, prikazu i kontroleru. Model se brine o postavljanja upita bazi. Prikaz (engl. *View*) koristi se za prikaz podataka u aplikaciji. Kontroler služi za upravljanje interakcije između aplikacije i korisnika. MVT obrazac je Model-Prikaz-Predložak (engl. *Model-View-Template*) i drugačiji je koncept od MVC-a. Razlika između MVC i MVT je kako se pristupa kontroleru i prikazu. Kod MVC kontroler je poseban entitet koji je razvijen od strane programera, dok kod MVT-a Django sam preuzima ulogu kontrolera, a programer se fokusira na modele, prikaze i predloške kao što je prikazano na slici.



Slika 2.1 Prikaz rada u Django okviru

Na slici 2.1 je vidljivo kako korisnik traži resurs za Django, on djeluje kao kontroler i provjerava dostupne resurse u URL-u. Ako se URL mapira, poziva se prikaz koji ide u interakciju s modelom ili predloškom. Nakon toga Django odgovara korisniku i šalje odgovor.

Određene prednosti Djanga su brzina razvoja, jer Django dolazi s već ugrađenim funkcionalnostima kao što su autentifikacija, upravljanje sesijama. Uz brzinu, prednost je i sigurnost zato što Django sam brine o sigurnosnim aspektima kao zaštita od SQL injekcija i slično. No Django dolazi i s određenim manama. Iako Django nudi fleksibilnost, monolitna struktura koju ima može postati neefikasna kod velikih aplikacija. Među mane Djanga možemo navesti i performanse zato što aplikacije znaju biti vrlo zahtjevne.

Django je moćan okvir koji omogućava brz i siguran razvoj web aplikacija koristeći Python. Fleksibilnost, sigurnost i ugrađene funkcionalnosti čine ga izvrsnim izborom za razvoj određenih projekata.

2.2 Ruby on Rails

Ruby on Rails često se naziva i samo Rails, popularan je razvojni okvir za web aplikacije nastao na programskom jeziku Ruby. Razvijen je s ciljem da pojednostavi i ubrza razvoj web aplikacija. U ovom okviru koriste se principi konvencije o konfiguraciji (engl. *Convention over Configuration*) i ne ponavljaj svoj kod (engl. *Don't Repeat Yourself*) koji omogućuju programerima da pišu što manje koda, a održavaju visoku produktivnost. Rails sa sobom nosi alate i module za rad s bazama podataka, upravljanje sesijama i autentifikaciju korisnika i to uvelike smanjuje potrebu za dodatnim vanjskim bibliotekama.

Rails se drži MVC strukture i svaki dio strukture ima svoju ulogu. Modeli su sloj koji radi s bazom podataka, a pogled (engl. *View*) je sloj koji se sastoji od predložaka zaslužnih

za prikaz podataka aplikacije. Za poglede uobičajeno se koriste ERB datoteke, koje su zapravo HTML datoteke proširene Ruby kodom. Kontroleri su sloj koji se bavi obradom HTTP zahtjeva.

Rails omogućava automatsko generiranje koda putem generatora koji stvara osnovne datoteke za modele, kontrolere i poglede. Osim toga Rails koristi migracije za jednostavno upravljanje promjenama na bazi. S aspekta sigurnosti Rails dolazi s ugrađenim sigurnosnim mehanizmima kao što su zaštita od SQL injekcija, XSS (engl. *cross-site scripting*).

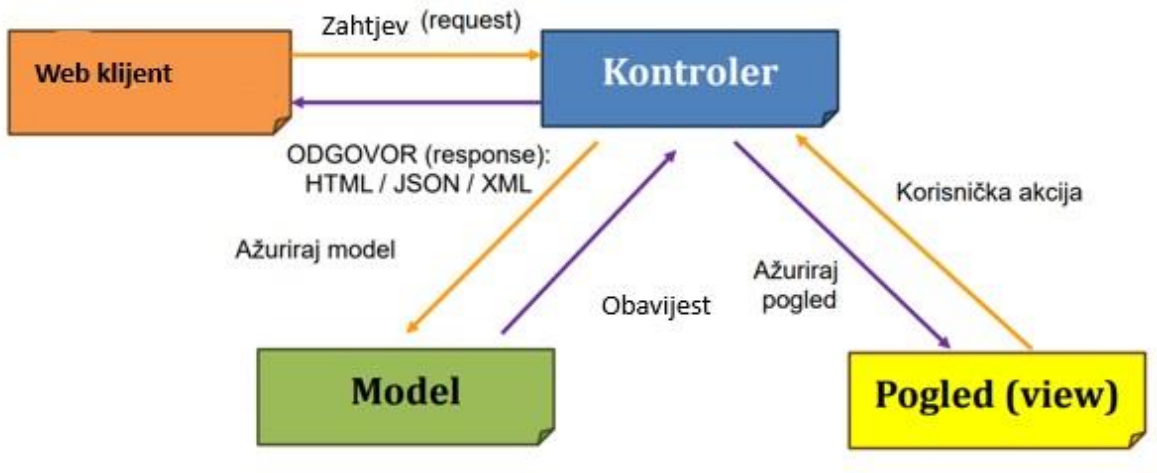
Prednosti razvoja u razvojnom okviru Ruby on Rails su brzina razvoja aplikacija zbog unaprijed definiranih struktura i pravila čime se izbjegava pisanje dodatnog koda. Velika prednost Railsa je velika zajednica korisnika i alata koji svojim dodacima za rad u ovom razvojnom okviru programerima uvelike olakšavaju rad, dok nedostatci Railsa su sporije performanse kod velikih aplikacija, monolitna arhitektura i drugi. Problem monolitne arhitekture može otežati skaliranje i upravljanje u dugoročnom razvoju aplikacija, posebno ako aplikacije postanu kompleksnije.

Ruby on Rails najčešće se koristi u startup projektima zbog svoje brzine razvoja zato što omogućava brzu izradu aplikacija s minimalnim naporom. Jedan poznati primjer primjene Ruby on Rails je GitHub. Platforma koja služi za upravljanje verzijama koda te omogućava programerima suradnju na projektima. GitHub Rails koristi se za backend, zato što omogućava brzo i efikasno upravljanje velikim brojem korisnika i podataka.

2.3 Spring

Spring je među najpopularnijim razvojnim okvirima za Java aplikacije, poznat po fleksibilnosti i robusnosti. Spring olakšava izgradnju skalabilnih i održivih sustava koristeći principe poput inverzije kontrole i aspektno orijentiranog programiranja. Springova arhitektura omogućuje programerima da biraju samo one komponente koje su im potrebne te se time smanjuje složenost koda. Unutar Springa postoji Spring Boot koji je dio šireg Spring ekosustava. Spring Boot pojednostavljuje razvoj mikroservisa te ga čini popularnim izborom za cloud-native aplikacije.

Što se tiče arhitekture, Spring aplikacija sastoji se od poslužiteljskog dijela aplikacije i klijentskog dijela. Klijentski dio je prilično jednostavan i sastoji se samo od pogleda (views), dok je poslužiteljskog dio kompleksniji i sastoji se od kontrolera, modela, servisa, entiteta, repozitorija i baze podataka. Prikaz rada Spring aplikacije temeljene na MVC prikazan je na slici 2.3.



Slika 2.3 Prikaz rada Spring aplikacije

Slika opisuje rad aplikacije gdje klijent šalje zahtjev prema kontroleru. Kontroler ima ulogu primanja korisničkih zahtjeva. On određuje taj zahtjev i odlučuje koji će dio poslovne logike ili podaci biti obrađeni. Također, kontroler odlučuje koji model treba ažurirati ili koji će podatak poslati prikazu. Model predstavlja poslovnu logiku aplikacije i inače radi s podacima iz baze podataka. Nakon obrade podataka u modelu, kontroler odlučuje kako ažurirati prikaz. Prikaz je odgovoran kako će podaci biti prikazani korisniku.

Uvođenje Spring Boota u Spring aplikacije omogućio je brži razvoj aplikacija, da osigura veći broj nefunkcionalnih zahtjeva kao što su sigurnost aplikacije, ugrađeni serveri i drugi.

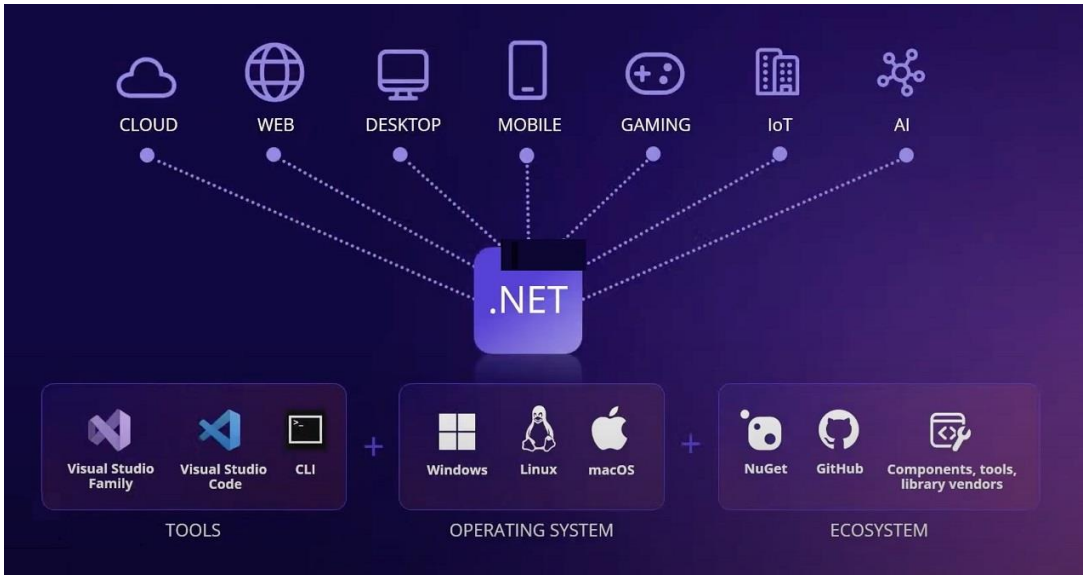
Modularnost i fleksibilnost velika su prednost Springa zato što nudi veliki broj modula koji omogućuju programerima da koriste samo one komponente koje su im stvarno potrebne, što Spring čini fleksibilnim za različite vrste projekata. Također, prednost Springa je poboljšana podrška za transakcije. Samim tim Spring se često koristi u aplikacijama koje koriste mikroservise, čime se omogućava stabilno i sigurno izvršavanje aplikacija. No kako svaka tehnologija i razvojni okvir donosi sa sobom neke prednosti, s njim dolaze i mane. Spring nije jednostavan razvojni okvir te samim tim za učenje je potrebno dosta vremena. Također, korištenje Springa može biti kompliciranо za manje projekte zato što njegova složenost i broj modula mogu biti preveliki za manje projekte.

Tako se može zaključiti da je Spring najbolje koristiti u aplikacijama za bankarstvo i financije. To mogu primjerice biti aplikacije za online plaćanje, transakcije ili čak sustavi za borbu protiv prevara.

2.4 .NET

.NET je razvojni okvir tvrtke Microsoft, koji omogućava razvoj aplikacija za web, mobilne aplikacije, desktop i cloud. Kao razvojni okvir na strani poslužitelja, .NET pruža podršku za visoko učinkovite i sigurne web aplikacije.

Što se tiče arhitekture .NET platforma dizajnirana je da podržava razvoj različitih vrsta aplikacija. Glavni jezici koje .NET podržava je C#, F# i VB.NET. Zahvaljujući zajedničkom CLR, unutar istih projekata moguće je koristiti više različitih jezika. CLR je *Common Language Runtime,* i predstavlja osnovni dio .NET arhitekture koji omogućuje izvršavanje aplikacija. CLR upravlja memorijom, vrši upravljanje iznimkama i sigurnošću. Unutar .NET arhitekture ne smije se izostaviti BCL standardna knjižnica koja pruža osnovne funkcionalnosti i komponente kao što su rad s datotekama. Razvojni okviri unutar .NET-a su ASP.NET Core namijenjen izradi modernih web aplikacija i API-ja, Xamarin koji se koristi za izradu mobilnih aplikacija i Windows Forms okvir za razvoj desktop aplikacija. .NET Core je cross platform varijanta .NET-a koja omogućava izradu aplikacija koje mogu raditi na operativnim sustavima Windowsu, macOS- u i Linuxu, s fokusom na performanse i skalabilnost. .NET koristi sustav paketa NuGet, s pomoću kojeg je unutar projekta moguće dodati vanjske biblioteke i module čime se olakšava proširenje funkcionalnosti aplikacija bez pisanja dodatnog koda. Slika 2.4 prikazuje alate, sustave i ekosisteme koji pomažu razvoju .NET aplikacija



Slika 2.4 Alati, sustavi i ekosistemi koji pomažu razvoju .NET aplikacija

Prednosti .NET-a su više platformska podrška (engl. *Cross-Platform*), visoke performanse i snažna sigurnost, dok su mane velika memorijska potrošnja zato što .NET aplikacije zahtijevaju znatne resurse i ograničena podrška za starije tehnologije. Primjena .NET-a kao što je već navedeno pokriva razvoj od web do mobilnih aplikacija, dok za stvarni primjer primjene .NET tehnologije možemo navesti Stack Overflow web aplikaciju. To je platforma koja omogućuje razmjenu znanja među programerima i inženjerima. ASP.NET Core osigurava skalabilnost i visoke performanse, što je važno s obzirom na ogroman broj korisnika i zahtjeva na platformi.

3. Razvojni okviri na strani klijenta

Razvojni okviri na strani klijenta glavni su alat za razvoj modernih web aplikacija. Pomažu stvaranje dinamičkih i responzivnih aplikacija koje se izvršavaju u korisnikovom pregledniku. Prednost ovih alata je što programerima omogućuju razvoj aplikacija s modularnim i ponovno upotrebljivim kodom. Za razvoj aplikacija kao razvojni okviri na strani klijenta koriste se Angular, React, Vue.js, jQuery i Blazor. Ovi razvojni okviri olakšavaju razmjenu podataka između klijentske i poslužiteljske strane i koriste tehnike za stvaranje jednostraničnih aplikacija (engl. Single Page Aplikacija - SPA). U jednostraničnim aplikacijama stranice se ažuriraju bez ponovnog učitavanja cijelog sadržaja stranice što programerima pruža lakše održavanje aplikacija, ali glavna mana ovih razvojnih okvira je složenost učenja.

3.1 Angular

Jedan od najpopularnijih JavaScript okvira (engl. Framework) za razvoj dinamičkih i interaktivnih web aplikacija je Angular. Razvijen je od strane Googlea i pruža platformu za izgradnju aplikacija koje se izvršavaju na klijentskoj strani. Pomoću modularnosti i dodavanjem raznih alata unutar ekosustava, Angular je omogućio stvaranje visoko responzivnih aplikacija uz jednostavno održavanje. Angular dozvoljava rad sa standardnim HTML-om i CSS-om, ali proširuje mogućnosti HTML-a. Sposobnost za izgradnju kompleksnih sučelja čini Angular idealnim izborom za jednostranične aplikacije (engl. Single Page Aplikacija - SPA), a za to je ključno da se interakcija odvija bez ponovnog učitavanja stranice.

Angular ne pokriva rad na strani poslužitelja tako da se uz Angular mora koristiti još neka tehnologija. Primjerice, za razvoj poslužiteljske strane projekta može se koristiti Java, Ruby, C# i drugi. „Ukratko, Angular je sve ono što bi HTML trebao biti, da je razvijen za razvoj aplikacija. HTML je odličan deklarativni jezik za statičke dokumente.“[5] HTML-om se ne može razvijati logika i za to se koriste okviri kao Angular i React. Razvoj aplikacija se pojednostavljuje pomoću Angulara tako što daje apstrakciju programerima. Angular nije idealna opcija za razvoj aplikacija, ali za CRUD (Create, Read, Update, Delete) aplikacije je više nego dobar, dok za razvoj igara i GUI editora i nije baš najbolji.

Arhitektura Angulara predstavlja moderni i modularni razvoj web aplikacija, koje se oslanjaju na neke ključne komponente koje omogućuju izgradnju održivih sustava. Središte

arhitekture čine komponente, koje su osnovni građevni blokovi aplikacije. Svaka se komponenta sastoji od HTML predložaka, CSS za stil i TypeScript koda za određivanje logike. Komponente omogućuju stvaranje ponovno upotrebljivih dijelova korisničkog sučelja te se time olakšava upravljanje složenim aplikacijama. Moduli su ključna uloga u Angular arhitekturi. Pomoću modula grupiraju se povezane komponente u funkcionalne cjeline. To omogućuje bolju organizaciju koda. Angular aplikacija mora imati glavni root modul te se u njemu povezuju svi ostali dijelovi aplikacije. Takva arhitektura optimizira performanse, a to se postiže tako da se određeni dijelovi stranice učitavaju tek kada su potrebni. Servisi su bitan dio arhitekture. Servisi omogućuju dijeljenje podataka i logike među različitim komponentama aplikacije. Pipes predstavljaju još jedan koristan alat u Angularu. Pipes se koristi za transformaciju podataka u predlošcima. Za formatiranje datuma, teksta, broja koriste se pipe bez potrebe za promjenom osnovnog koda komponente. Za kraj o arhitekturi Angulara može se reći da pruža robusnu i fleksibilnu platformu za razvoj dinamičnih web aplikacija.

Modularnost Angulara je velika prednost ovog razvojnog okvira koji omogućuju organizaciju koda na jasan način te to olakšava održavanje i ponovnu upotrebu koda. Mana Angulara je strma krivulja učenja. Angular je složen i ima veliko bogatstvo funkcionalnosti koje mogu izazvati probleme kod novih korisnika. Također, performanse se mogu ubrajati u mane, zato što kod većih aplikacija s puno podataka, Angular može postati sporiji.

Primjena Angulara u svijetu je velika. Jedan od primjera korištenja Angulara je Google Cloud Console. To je platforma koja omogućuje korisnicima da upravljaju svojim projektima. Angular se također koristi kod e-learning platformi. Takve aplikacije imaju različite module za registraciju korisnika, pregled dostupnih tečajeva i polaganje testova.

Modularnost i dvosmjerno povezivanje podataka kod Angulara olakšavaju razvoj složenih i interaktivnih web aplikacija . Također ima snažan ekosustav i podršku za optimizaciju koji su neki od razloga zašto koristiti Angular.

3.2 React

React je nastao 2013. godine kao JavaScript biblioteka za izradu korisničkog sučelja. Cilj je bio riješiti problem kompleksnosti izrade sučelja za web aplikacije. Stvoren je od strane programera Facebooka. React pruža jednostavnost i efikasnost, a i lako se integrira s drugim tehnologijama poput Node.js i Redux. Može se koristiti kao baza za projekte jednostraničnih ili progresivnih web aplikacija. Većinom glavna uloga ovog razvojnog okvira je prenošenje podataka modelu objekta dokumenta. React nije samo za razvoj web

aplikacija nego ih je moguće i kreirati za mobilne uređaje pomoću React-Native. Zahvaljujući tome, sve te karakteristike omogućavaju programerima da razviju velike web aplikacije s mogućnosti promjene podataka na stranici bez potrebe za osvježavanjem stranice. To omogućava da React aplikacije budu brze, jednostavne i skalabilne.

Arhitektura Reacta se temelji na komponentama koje imaju svoje stanje i mogu prikazivati korisničko sučelje na temelju tog stanja. Ključni element Reacta je virtualni DOM. To omogućava da React prvo izračunava promjene u virtualnom modelu pa primjenjuje samo nužne izmjene. Kod arhitekture naglasak je na izgradnji korisničkih sučelja što znači da programeri opisuju što žele prikazati, a ne kako da se to postigne. Takva arhitektura pojednostavljuje kod i čini ga lakim za održavanje i čitanje. Ono što React čini fleksibilnim i modularnim za gradnju sučelja je mogućnost prijenosa podataka između komponenti. Bogati ekosustav biblioteka React čini pogodnim za integraciju u različitim tehnologijama i alatima.

Kao i svaki razvojni okvir, i React ima određene prednosti zašto ga koristiti. Jednostavnost uporabe i učenje velika su prednost za privlačenje novih korisnika. „Reactov pristup utemeljen na komponentama i upotrebom jednostavnog JavaScript-a pojednostavljuju izradu profesionalnih mobilnih i web aplikacija.“[5] No to nije najbolji izbor za početnike, jer React zahtijeva znanje u JavaScriptu, CSS-u i HTML-u. Prednost React aplikacija su dobre performanse tih aplikacija. No nekad su problem Reacta česte nadogradnje što može uzrokovati probleme s kompatibilnošću.

U primjeni, React je dobar primjer korištenja nekog razvojnog okvira za razvoj online trgovine. Komponente kao kartice proizvoda, košarice i druge bile bi modularno organizirane, kao i prikazom stanja proizvoda. Kao neki stvarni primjer primjene Reacta najbolje je spomenuti Facebook. React kod Facebook-a pruža ažuriranje feedova, komentara i drugih interaktivnih elemenata bez ponovnog učitavanja cijele stranice.

3.3 jQuery

jQuery je popularni JavaScript okvir razvijen s ciljem olakšavanja i ubrzavanja rada JavaScriptom, manipulacije HTML dokumentima, upravljanja događajima i asinkronim pozivima. Jednostavniji je nego osnovni JavaScript. jQuery sintaksa omogućila je brže pisanje i čitanje koda. Značajno je olakšan rad s asinkronim JavaScriptom (engl. *Asynchronous JavaScript And XML - AJAX*) te su time web aplikacije dinamičnije i responzivnije.

Arhitektura se temelji na funkcionalnom pristupu. CSS selektori omogućuju jednostavan odabir HTML elemenata. Značajno je pojednostavljeno upravljanje događajima na elementima Web stranica. Za upravljanje događajima jQuery koriti drugačiju sintaksu od standardnog JavaScripta. Arhitektura jQueryja nudi jednostavnu implementaciju animacija i efekata nad HTML elementima. Taj sustav animacija je optimiziran te nudi dobre performanse na aplikacijama. jQuery nudi proširenja funkcionalnosti putem dodataka. Oni nude programerima dodatne funkcionalnosti poput galerija slika, kalendara ili složenih korisničkih interakcija. Jedna od najbitnijih komponenti arhitekture je sposobnost da se normalizira ponašanje JavaScripta između različitih web preglednika. Arhitektura je građena modularnim pristupom koji omogućuje da se brzo i efikasno grade dinamičke web aplikacije.

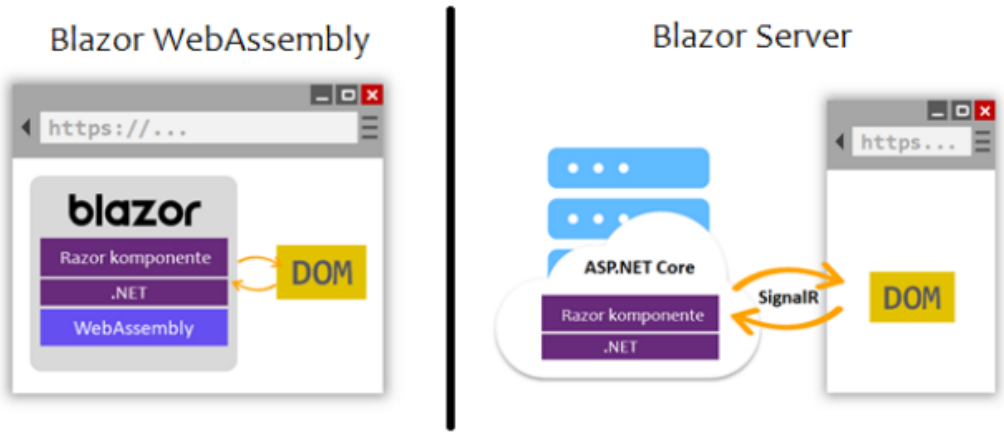
Kao i svaki razvojni okvir i ovaj ima svoje prednosti i mane. Prednosti jQueryja su jednostavnost korištenja, kompatibilnost na svim pretraživačima i bogata biblioteka dodataka. Dok su prednosti opisane kroz arhitekturu samog razvojnog okvira, mane jQueryja su manjak modularnosti, smanjene performanse kod velikih aplikacija i zastarjelost. Ne pruža se dovoljna razina modularnosti kao moderni razvojni okviri. Zbog smanjenih performansi kod velikih aplikacija ovi projekti znaju biti spori i ne efikasni. Također jQuery je danas zastario i zamijenili su ga mnogi noviji i moderniji okviri.

jQuery trenutno je najbolji za manje projekte. Često se koristi za dodavanje jednostavne dinamike web stranicama. Danas se jQuery najviše koristi za neke projekte koji su zastarjeli i na njima je teško implementirati nove i moderne razvojne okvire. Kao primjer gdje se još uvijek danas koristi jQuery je WordPress, iako se WordPress razvija kao platforma koristeći razne tehnologije, jQuery je i dalje ključan za rad.

3.4 Blazor

Blazor je razvojno okruženje koje omogućuje izgradnju web aplikacija pomoću C# programskog jezika umjesto JavaScript-a. „Blazor koristi tehnologiju WebAssembly, što omogućuje da se C# kod izvodi u pregledniku bez potrebe za posebnim runtime-om ili pluginom.“[6] Ovaj razvojni okvir korisniku daje mogućnost razvoj aplikacije i sa serverske strane i klijentske. WebAssembly značajno smanjuje vrijeme za razvoj aplikacije, zato što se jedan programski jezik i jedna tehnologija koriste za razvoj na korisničkoj i serverskoj strani. U Blazoru se također mogu koristiti sve biblioteke i alatke unutar .NET ekosustava. Blazor aplikacije također se mogu koristiti drugim .NET tehnologijama kao što su ASP.NET i Entity Framework, što olakšava razvoj.

Unutar samog Blazora postoje dva oblika izvođenja. Blazor WebAssembly i Blazor Sever, razlika među njima je lokacija gdje se izvršava kod. Na slici 3.1 prikazano je kako se Blazor WebAssembly izvodi u samom klijentskom pregledniku, dok Blazor Server se izvodi na poslužitelju te promjene šalje u preglednik.



Slika 3.1 Prikaz izvođenja koda u Blazoru

Takva arhitektura za Blazor Server i WebAssembly sa sobom nosi prednosti i nedostatke. Kod Blazor Servera prednost je što korisnik ne mora preuzimati cijelu aplikaciju nego poslužitelj obrađuje teške zadatke, dok je nedostatak što korisnik mora imati stalni pristup internetu i postoji mogućnost latencije zbog slanja zahtjeva poslužitelju. Blazor Assembly isto tako ima i prednosti i mane. Prednost je to što aplikacije rade bez interneta i smanjuje se opterećenje na poslužitelju, a nedostatak je veće vrijeme učitavanja pošto se cijela aplikacija mora preuzeti u preglednik, a resursi mogu biti ograničeni. Tako Blazor omogućuje izradu modernih web aplikacija bez korištenja JavaScripta, s različitim opcijama arhitekture ovisno o potrebama aplikacije. No kada se Blazor gleda kao jednu cjelinu, njegova najveća prednost je razvoj u C# programskom jeziku što omogućava dijeljenje koda između klijenta i poslužitelja. Zato je mana Blazora to što je ograničen gotovim komponentama. U usporedbi sa Reactom ili Angularom, Blazor još nema tako bogat katalog gotovih komponenti i datoteka.

Što se tiče primjene Blazora uglavnom su to interni poslovni sustavi ili dashboard aplikacije. Primjerice, kod *dashboard* aplikacija Blazor je idealan, jer nudi prikaz podataka u stvarnom vremenu. Na primjer, to su financijska nadzorna ploča(engl. *Dashboard*) ili alati za prikaz performansi. Blazor je sve popularniji zbog mogućnosti razvoja *full-stack* aplikacija u C# programskom jeziku.

4. Relacijski model baze podataka

Relacijski model baze podatke je jedan od najčešće korištenih modela za organizaciju i upravljanje podacima. Temelji se na matematičkom konceptu relacija. Te relacije se koriste za predstavljanje podataka i njihovih odnosa. U relacijskom modelu baze podataka, u svakom tipu podataka svi se podaci spremaju tablično. Svakom tipu podataka odgovara jedna tablica. Svaka tablica predstavlja entitet. Redovi unutar te tablice su zapisi tog entiteta, dok atributi odnosno stupci opisuju te zapise.

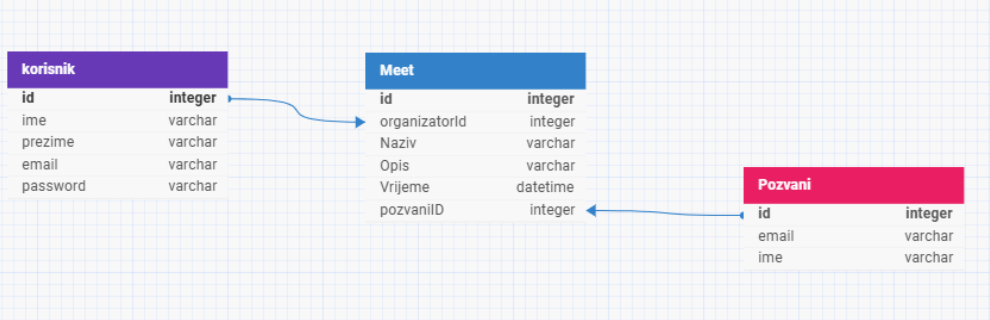
U ovom modelu nisu bitni ni redoslijed zapisa ni polja. Relacije među tablicama uspostavljaju se preko primarnih ključeva. Primarni ključevi osiguravaju jedinstvenost svakog zapisa u tablici. Strani ključevi povezuju različite tablice i omogućuju odnos među tablicama. Ovakav model baze podataka postiže fleksibilnost u organizaciji podataka. Normalizacija je ključni proces za dizajn relacijskih baza podataka čime se osigurava dosljednost podataka. U suštini ovaj proces se koristi kako bi se podaci organizirali na učinkovit način.

Prednosti ovog modela baze podataka su jednostavnost upotrebe, integritet podataka i široka upotreba u industriji.

4.1 Strukturiranje modela baze

Strukturiranje baze podataka ključni je korak u dizajnu sustava za upravljanje podacima. Na taj način se određuje kako se podaci organiziraju, pohranjuju i kako im se pristupa. Pravilnim strukturiranjem modela baze osigurava efikasno upravljanje velikim količinama podataka. Relacijski model je među najkorištenijim modelima za strukturiranje baze.

Za stvaranje dobre strukture za bazu podataka prvo je potrebno kreirati koncept modela baze. Koncepti se najčešće izrađuju pomoću dijagrama entitet-veza. Takav dijagram prikazuje entitete u sustavu i atribute te veze među njima. Entiteti predstavljaju stvarni objekt primjerice „Klijent“ ili „Proizvod“, dok atributi opisuju taj entitet. Odnosi među entitetima ključan su dio koncepta, jer definiraju kako su podaci međusobno povezani. Relacije mogu biti različitih vrsta. Na slici 4.1 prikazan je jednostavan primjer relacija među entitetima.



Slika 4.1 prikaz relacija među entitetima.

Slika 4.1 prikazuje kako entitet Sastanak (engl. *Meet*) prima podatke o organizatoru sastanka iz entiteta Korisnik, a o pozvanima iz entiteta Pozvani.

Nakon konceptualnog modeliranja potrebno je izraditi logički model baze podataka. U ovom koraku sve što je konstruirano u konceptu treba posložiti logički. Unutar ovog koraka svaki entitet postaje tablica, dok svaki atribut postaje stupac unutar nje. Logički model uključuje definiranje primarnih ključeva tablica. Primarni ključevi osiguravaju jedinstvenost svakog zapisa u tablici. Unutar tablica imamo i strane ključeve koji omogućavaju relacije među tablicama. Ključan dio logičkog modela je normalizacija i taj proces eliminira redundantne podatke i osigurava integritet baze podataka. Normalizacija također sprječava anomalije prilikom umetanja, brisanja ili ažuriranja podataka.

Treći korak je fizički model baze podataka i on se bavi načinom na koji će se podaci fizički pohranjivati. U ovoj fazi je ključno razmotriti tehničke zahtjeve, primjerice performanse sustava, potrebna brzina za odgovor i dostupnost prostora za pohranu. Fizički model također obuhvaća definiranje indeksa na stupcima kao primarni ključ i strani ključ. Time se ubrzava pretraživanje i pristup podacima.

Bitno je za strukturiranje modela baze održati njen integritet podataka. Modeli baza podataka koriste se u raznim domenama, od financijskih sustava do internet trgovine i društvenih mreža.

4.2 Normalizacija baza podataka

Normalizacija je postupak organizacije podataka u bazi podataka. Koristi se kako bi se smanjilo ponavljanje podataka i da bi se osigurala konzistentnost podataka. Cilj je stvoriti učinkovitu bazu podataka u kojoj se podaci pravilno organiziraju, eliminiraju duplicirani podaci i da se osigurava integritet. Normalizacija se provodi kroz različite razine i to nazivamo normalne forme (NF).

Prva normalna forma postavlja zahtjeve za organizaciju podataka u tablici. Kako bi podaci u tablici bili 1NF svi zapisi unutar nje moraju biti jedinstveni. Primjer tablice gdje se ne poštuje 1NF prikazan je u tablici 4.1.

Tablica 4.1 Prikaz tablice koja nije 1NF

ID	Ime	Prezime	Telefon
1	Ivan	Ivić	091-123456, 099-654321
2	Ana	Anić	095-111111

Tablica koja poštuje 1NF ne bi trebala imati više podataka u jednom stupcu. U ovom primjeru vidimo kako stupac „Telefon“ sadrži više od jednog korisnika. Takva tablica nakon normalizacije izgledala bi kao tablica 4.2.

Tablica 4.2 Prikaz tablice nakon 1NF

ID	Ime	Prezime	Telefon
1	Ivan	Ivić	091-123456
2	Ivan	Ivić	099-654321
3	Ana	Anić	095-111111

Kako bi tablica bila u drugoj formi mora se ispuniti 1NF. Kako radi druga normalna forma(2NF) najbolje je objasniti na primjeru.

Tablica 4.3 Prikaz tablice koja nije unutar 2NF

NarudžbaID	ProizvodID	ImeProizvoda	Količina
1	101	Laptop	2
1	102	Miš	5
1	101	Laptop	1

U tablici 4.3 atribut „ImeProizvoda“ ovisi samo o „ProizvodID“, a ne o cijelom primarnom ključu. Normalizacija bi ovu tablicu podijelila na dvije tablice. Dobili bi tablicu 4.4 koja bi bila tablica narudžba.

Tablica 4.4 Tablica narudžba

NarudžbaID	ProizvodID	Količina
1	101	2
1	102	5
2	101	1

Druga tablica 4.5 koja bi nastala nakon normalizacije bila bi tablica Proizvod.

Tablica 4.5 Tablica proizvod

ProizvodID	ImeProizvoda
101	Laptop
102	Miš

Nakon druge normalne forme dolazi na red treća. Kako bi se mogla izvršiti treća normalna forma potrebno je ispuniti uvjete 2NF. Zadatak treće normalne forme da ni jedan ne ključni atribut ne smije ovisiti o nekom ne ključnom atributu. Drugačije rečeno svi ne ključni atributi moraju izravno ovisiti o primarnom ključu.

Prednosti normalizacije su smanjenje redundancije, povećanje integriteta, fleksibilnost i skalabilnost. Organizirani i povezani podaci olakšavaju dodavanje novih podataka i povezivanje tablica, dok su nedostatci složenije postavljanje upita, performanse i dodatno vrijeme razvoja. Normalizacija može usporiti aplikacije zbog većeg broja pridruživanja između tablica.

4.3 Sigurnost baza podataka

Sigurnost baza podataka odnosi se na metode i mjere zaštite baza podataka od neovlaštenog pristupa, gubitaka podataka i oštećenja povjerljivih informacija. Razvoj tehnologije dovodi do porasta napada pa tako i sigurnosne mjere postale su neophodne za zaštitu podataka.

Sigurnost baza podataka suočava se s puno različitih prijetnji. Prijetnje mogu biti razne no jedna od najčešćih je neovlašteni pristup. Neovlašteni pristup je prijetnja gdje napadači koriste ranjivost u sustavu kako bi pristupili podacima. To često rezultira krađom informacija. SQL injekcije su također česta vrsta napada. U ovoj vrsti napada napadači najčešće pokušavaju umetanje SQL naredbi unutar poziva prema bazi iz web aplikacije.

Zlonamjerni zaposlenici također mogu biti prijetnja. Ti korisnici, ako imaju lošu namjeru, uz pristup bazi mogu uzrokovati ozbiljnu štetu.

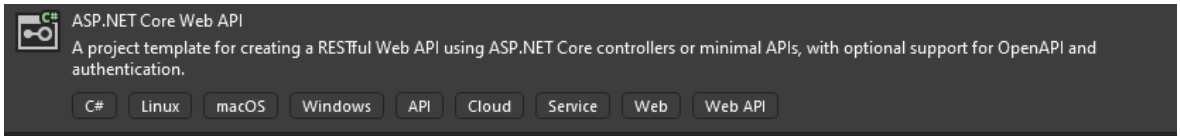
Kako bi osigurali svoju bazu potrebno je napraviti kontrolu pristupa. Kontrola pristupa ograničava korisnike na njihov pristup podacima u bazi i operacije koje mogu obavljati. Dodatna enkripcija također nije loš način za zaštitu. Podaci koji su pohranjeni u bazi podataka i podaci koji se prenose između baze i aplikacije mogu biti šifrirani kako bi se osigurala sigurnost tih podataka i mogu im pristupiti samo korisnici koji su ovlašteni za to. Jedna od važnijih sigurnosnih mjera je redovno ažuriranje baze i softvera aplikacije. Samim ažuriranjem osiguravaju se najnovije sigurnosne zakrpe. Sigurnosne kopije podataka pomažu u sigurnosti i očuvanju podataka. Stvaranjem sigurnosnih kopija osigurava se zaštita u slučaju kvara, napada ili ljudske greške na bazi.

Zaključak koji se treba donijeti u vezi sigurnosti baze podataka je da je to živ proces i treba ga stalno ažurirati i održavati. U svijetu gdje su podaci svakim danom sve bitniji i vrjedniji, potrebno je osigurati bazu podataka i dodatno zaštiti te informacije.

5. Razvoj web aplikacije

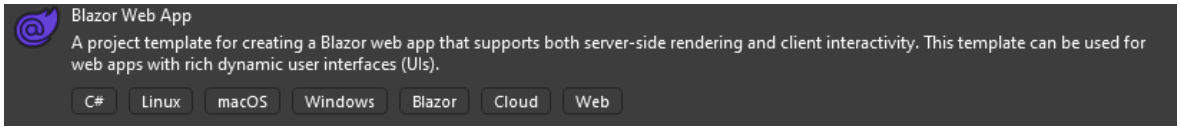
Cilj ovog završnog rada bila je izrada web aplikacije za dogovaranje sastanaka pomoću tehnologija Blazor i .NET API te upoznavanje s njima, uočavanje određenih prednosti i nedostataka. Primjer aplikacije koji je izrađen u nastavku rada mogao bi se koristiti u zatvorenim sustavima određenih kompanija ili za javnu upotrebu uz određene dorade. Aplikacija uz Blazor i .NET API kao vanjski servis za komunikaciju koristi Gmail. Korisnik bi u ovoj aplikaciji mogao kreirati poziv za sastanak te bi pozvani korisnik putem Gmaila mogao potvrditi svoj dolazak ili ga odbiti.

Aplikacija je razvijana u Visual Studio 2022 u C# programskom jeziku. Kako je za izradu aplikacije trebalo koristiti Blazor i .NET API, kreirana su dva projekta unutar jednog *solutiona*. Projekt koji se bavi povezivanjem s bazom podataka i razvojem logike same aplikacije je .NET API. Na slici 5.1 prikazana je opcija koja je korištena unutar Visual Studia kako bi se kreirao projekt.



Slika 5.1 Odabrani predložak za API

Nakon toga unutar istog *solutiona* kreiran je i projekt za izradu korisničkog dijela aplikacije. Za izradu korisničkog dijela korišten je Blazor i na slici 5.2 prikazan je predložak koji je korišten.



Slika 5.2 Odabrani predložak za Blazor

Iako Blazor podržava rad i na korisničkoj i na poslužiteljskoj strani, za izradu ove aplikacije odabrane su dvije tehnologije iz razloga što je Blazor još uvijek relativno novija tehnologija na tržištu i zbog toga ne nudi najbolja rješenja za izradu aplikacije. Jedan od prvih problema koji se pojavio je što u trenutku početka izrade aplikacije Blazor u potpunosti nije podržavao sve mogućnosti .NET 8 verzije. Zbog toga i lakšeg povezivanja između projekata za izradu aplikacije korištena je verzija 7 .NET-a.

Nakon što su opisane određene pojedinosti oko stvaranja *solutiona* i projekata unutar istog, potrebno je pojasniti razvoj aplikacije. Prvi korak u samom razvoju bilo je kreiranje modela sastanka.

Programski kod 5.1: Stvaranje modela Meeting

```
public class Meeting
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public DateTime Date { get; set; }
    public string Time { get; set; }
    public string OrganizerEmail { get; set; }
    public List<Invitee> Invitees { get; set; } = new List<Invitee>();
}
```

Programski kod 5.1 prikazuje klasu koja predstavlja model Sastanka. Model sastanka sadrži određene atribute koji se će se kasnije koristiti u aplikaciji. „Id“ je jedinstveni element svakog sastanka koji će se kasnije generirati u bazi podataka. „Title“ je naslov koji će svaki sastanak nositi. „Description“ je opis svakog sastanka koji će pružati određene dodatne informacije o sastanku. „Date“ prikazuje datum kada će sastanak biti održan. „Time“ je vrijeme kada će se sastanak održati na odabrani datum. „OrganizerEmail“ je element ovog modela koji će u aplikaciji prikazivati informaciju o korisniku koji će organizirati sastanak, dok element „Invitees“ je lista sudionika koja se pohranjuje kao objekt tipa „Invitee“. Klasa Meeting pruža osnovnu strukturu za organizaciju sastanaka. Ovakav model olakšava rad s podacima sastanka, ali i pruža prilagodbe u slučaju promjene same ideje vezane za aplikaciju. Nakon klase Meeting kreirana je klasa Invitee.

Programski kod 5.2: Kreiranje modela Invitee

```
public class Invitee
{
    public int Id { get; set; }
    public string Email { get; set; }
    public InvitationStatus Status { get; set; } =     InvitationStatus.Pending;

    public static implicit operator string(Invitee v)
    {
        throw new NotImplementedException();
    }
```

```
}
```

Izgled klasa Invitee prikazan je u programskom kodu 5.2. Klasa Invitee predstavlja model pozvanih sudionika na sastanke. „Id“ kao i kod klase Meeting predstavlja jedinstveni element za pozvanu osobu koji će služiti kao primarni ključ u bazi podataka. „Email“ je adresa pozvane osobe koja služi kao glavni način za komunikaciju. I element „Status“ prikazuje trenutni status pozivnice za sastanak. Klase Invitee i Meeting su povezane u odnosu jedan prema više što znači da svaki sastanak može imati više pozvanih osoba i zato se podatci iz klase Invitee pohranjuju u listu u klasi Meeting. Što se tiče statusa pozivnice u klasi Invitee, ti statusi su definirani u enum-u InvitationStatus prikazan u programskom kodu 5.3.

Programski kod 5.3 Definiranje statusa pozivnice

```
public enum InvitationStatus
{
    Accepted,
    Declined,
    Pending
}
```

Ovim enumom definirani su statusi pozivnice hoće li korisnik prihvatiti poziv, odbiti ili će status ostati u čekanju.

Nakon što su kreirani modeli na kojima će biti kreirani sastanci, dalje je trebalo tim modelima dati neku funkciju pa je razvijen kontroler za te modele. Stvoren je kontroler MeetingsController, unutar kojeg su stvorene prvobitne funkcije za stvaranje sastanka, brisanje sastanka, kako bi se napravile promjene u sastanku i za dohvaćanje kreiranih sastanaka. Nakon toga, idući korak bio je povezati API projekta sa SQL bazom podataka Veleučilišta u Bjelovaru. Kako bi API bio povezan s bazom podataka bilo je potrebno instalirati dodatne pakete kao što su Microsoft.EntityFrameworkCore.SqlServer, Microsoft.EntityFrameworkCore.Tools i Microsoft.EntityFrameworkCore.Design. Po instalaciji dodatnih paketa, trebalo je dodati konekcijski string za bazu podataka. Također je potrebno napraviti klasu za kontekst baze podataka koja će naslijediti DbContext, a u klasi su definirani entiteti koji će se mapirati na tablice u bazi. Kreiranjem klase za kontekst i dodavanjem konekcijskog *stringa* potrebno je registrirati ih u Program.cs kako bi aplikacija znala komunicirati s bazom. Nakon toga slijedi migracija i ažuriranje baze podataka. U tom trenutku u kontroler se može dodati mogućnost dodavanja i upravljanja podacima u bazi.

Dodavanjem kontroleru mogućnosti interakcije s bazom podataka može se dodatno i pojasniti funkcije unutar istog. Programski kod 5.4 prikazuje funkciju za čitanje svih sastanaka kreiranih unutar baze uključujući i pozvane sudionike što se osigurava korištenjem Include metode.

Programski kod 5.4 Funkcija za čitanje sastanaka

```
[HttpGet]
public async Task<IActionResult> GetAll()
{
    var meetings = await _context.Meetings.Include(m => m.Invitees).ToListAsync();
    return Ok(meetings);
}
```

Iduća funkcija kreira novi sastanak. Funkcija prima podatke o sastanku i dodaje ih u bazu, a nakon uspješne kreacije vraća informacije o sastanku, a kako to izgleda prikazano je u programskom kodu 5.5.

Programski kod 5.5 Funkcija za stvaranje sastanka

```
[[HttpPost]
public async Task<IActionResult> Create(Meeting meeting)
{
    _context.Meetings.Add(meeting);
    await _context.SaveChangesAsync();
    return CreatedAtAction(nameof(GetById), new { id = meeting.Id }, meeting);
}
```

Nakon što je dodana mogućnost kreiranja sastanka, bilo je potrebno dodati i mogućnost naknadnog uređivanja i izmjene samog sastanka. Unosom izmjena u funkciji Update prvo se provjerava postoji li navedeni sastanak u bazi te ako postoji sastanak se šalje u bazu. Kako radi funkcija Update prikazano je u programskom kodu 5.6.

Programski kod 5.6 Funkcija za ažuriranje podataka u vezi sastanka

```
[HttpPut("{id}")]
public async Task<IActionResult> Update(int id, Meeting meeting)
{
    var existingMeeting = await _context.Meetings.Include(m =>
m.Invitees).FirstOrDefaultAsync(m => m.Id == id);
    if (existingMeeting == null)
    {
        return NotFound();
    }
```

```
    existingMeeting.Title = meeting.Title;
    existingMeeting.Description = meeting.Description;
    existingMeeting.Date = meeting.Date;
    existingMeeting.Time = meeting.Time;
    existingMeeting.Invitees = meeting.Invitees;

    await _context.SaveChangesAsync();
    return NoContent();
}
```

Zadnja mogućnost za upravljanje sastancima je funkcija za brisanje. Ova funkcija briše sastanak iz baze na temelju njegovog ID-a. Programski kod 5.7 prikazuje funkciju za brisanje.

Programski kod 5.6 Funkcija za brisanje sastanka iz baze podataka

```
[HttpDelete("{id}")]
public async Task<IActionResult> Delete(int id)
{
    var meeting = await _context.Meetings.Include(m =>
m.Invitees).FirstOrDefaultAsync(m => m.Id == id);
    if (meeting == null)
    {
        return NotFound();
    }

    _context.Meetings.Remove(meeting);
    await _context.SaveChangesAsync();
    return NoContent();
}
```

Idući korak u razvoju bilo je dodavanje komunikacije e-poštom kako bi aplikacija mogla sudionicima slati pozivnice za sastanke. Da bi aplikacija mogla slati e-poštu potrebno je integrirati servis za slanje e-pošte. Servis za slanje e-pošte koristi standard za slanje e-pošte (engl. *Simple Mail Transfer Protocol SMTP*). Tako se u klasi EmailService stvara metoda za slanje e-pošte. Metoda SendEmailAsync prikazana je u programskom kodu 5.7.

Programski kod 5.7 Metoda za slanje e-pošte

```
public async Task SendEmailAsync(string to, string subject, string body, Meeting meeting)
{
   try
   {
      var fullBody = $"{body}<br/><br/>";

      var mailMessage = new MailMessage("filip0906@gmail.com", to, subject, fullBody)
      {
         IsBodyHtml = true
      };
      await _smtpClient.SendMailAsync(mailMessage);
      Console.WriteLine($"Email successfully sent to {to}");
   }
   catch (Exception ex)
   {
      Console.WriteLine($"Error sending email to {to}: {ex.Message}");
      throw;
   }
}
```

Nakon kreiranog servisa za slanje mailova u konstruktor kontrolera dodan je i servis za slanje e-pošte što daje mogućnost stvaranja funkcije koja će generirati poruku e-pošte. Funkcija SendEmailInvitations prikazana je u programskom kodu 5.8, a radi na način da prima podatke o nazivu sastanka, opisu, datumu i vremenu održavanja te dvije prilagođene poveznice za prihvatiti ili odbiti poziv na sastanak.

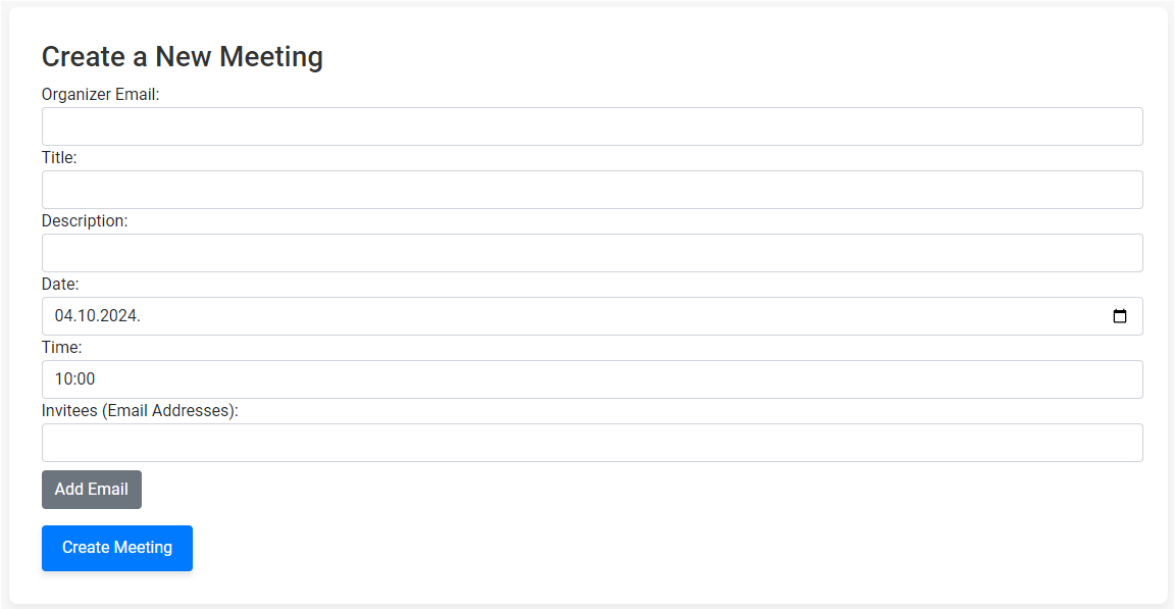
Programski kod 5.8 Funkcija koja kreira poruku e-pošte

```
[private async Task SendEmailInvitations(Meeting meeting)
{
   var baseUrl = $"{Request.Scheme}://{Request.Host}/"; // Generiraj bazni URL iz
HttpRequest objekta

   foreach (var invitee in meeting.Invitees)
   {
      var acceptUrl = $"{baseUrl}api/meetings/accept-invitation/{meeting.Id}/{invitee.Email}";
      var declineUrl = $"{baseUrl}api/meetings/decline-
invitation/{meeting.Id}/{invitee.Email}";

      var emailBody = $"You are invited to the meeting titled \"{meeting.Title}\" on
{meeting.Date.ToShortDateString()} at {meeting.Time}.<br/>" +
                  $"Description: {meeting.Description}<br/>" +
                  $"<a href='{acceptUrl}'>Accept Invitation</a><br/>" +
                  $"<a href='{declineUrl}'>Decline Invitation</a>";
```

```
        try
        {
            await _emailService.SendEmailAsync(invitee.Email, "Meeting Invitation", emailBody,
meeting);
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error sending email to {invitee.Email}: {ex.Message}");
        }
    }
}
```

Sada kada aplikacija ima mogućnost stvaranja poruka za e-poštu bilo je potrebno kreirati mogućnost za prihvatiti ili odbiti poziv za sastanak. Unutar kontrolera kreirane su poveznice s kojom će korisnik moći promijeniti status svog dolaska na sastanak. Obavijest o pozivu za sastanak slala bi se e-poštom nakon što je sastanak stvoren ili je na njemu napravljena izmjena čime su napravljene sve zamišljene funkcionalnosti na strani poslužitelja.

Idući korak u razvoju aplikacije bio je povezati Blazor web stranicu i njoj dodijeliti mogućnost upravljanja funkcionalnostima poslužiteljske strane. Za početak trebalo je namjestiti da se projekti API i Blazor pokreću zajedno te da se u Program.cs preko *localhosta* zajedno povežu. Za zadatak ovog dijela razvoja aplikacije bilo je potrebno kreirati stranice za kreiranje, uređivanje i pregled sastanaka. Prvo je kreirana stranica za kreiranje sastanaka uz što je dodan i prostor za unos podataka o sastanku. Također je dodana i validacija unesenih podataka, što znači da je obvezno unijeti ispravnu adresu e-pošte, naslov i opis sastanka te vrijeme kojeg korisnici biraju kada će se sastanak odvijati u budućnosti. Dodavanje sudionika na sastanak je odrađeno na način da se unesu njihove adrese e-pošte te nakon što su zadovoljeni svi uvjeti za kreiranje sastanka, podaci se šalju u API te se pohranjuju u bazu. Ako je sve dobro i zahtjev je prošao, korisniku se otvara prikaz njegovog sastanka. Način izgleda stranice za stvaranje sastanaka prikazan je na slici 5.3



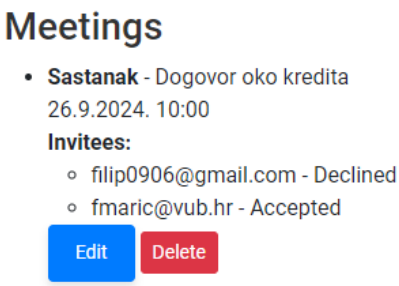
Slika 5.3 Stranica za stvaranje sastanaka

Nakon što je sastanak kreiran otvara se stranica sa sastancima. Kada se stranica sa sastancima učita šalje se GET zahtjev prema API-u kako bi dohvatio sastanke u bazi. Način na koji to funkcionira prikazano je u programskom kodu 5.9.

Programski kod 5.8 Metoda koja šalje GET zahtjev prema API-u

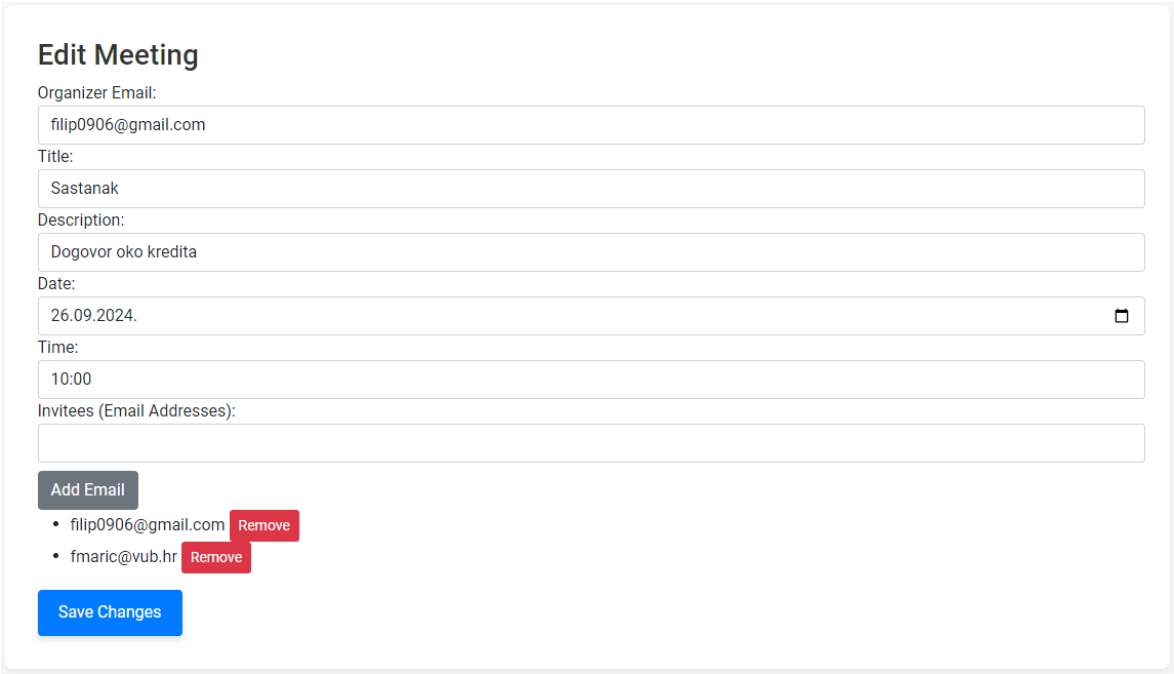
```
protected override async Task OnInitializedAsync()
{
    var client = ClientFactory.CreateClient("MeetingApi");

    try
    {
        meetings = await client.GetFromJsonAsync<List<Meeting>>("api/meetings");
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Error loading meetings: {ex.Message}");
    }
}
```

Svaki sastanak prikazan je u obliku liste koja uključuje naslov, opis, datum i vrijeme te popis sudionika i njihov status dolaska. Prikaz izgleda sastanka prikazan je na slici 5.4.



Slika 5.4 Kreirani sastanak

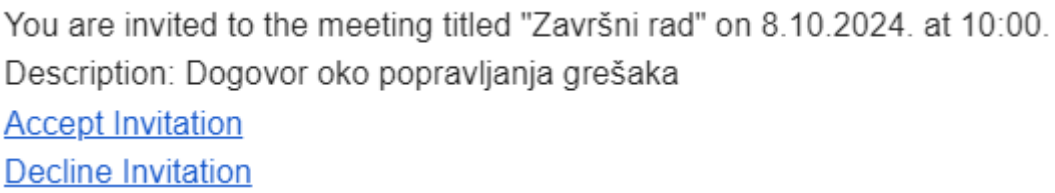
Kao što je vidljivo na slici 5.4 na ovom dijelu aplikacije korisnici imaju mogućnost brisanja sastanka i uređivanja sastanaka. Pritiskom na gumb Delete šalje se u API poziv za brisanje sastanka iz baze s ID sastanka, dok se pritiskom na gumb Edit otvara stranica za uređivanje sastanaka čijim otvaranjem se dohvaća ID sastanka koji treba urediti. Korisnik ima vidljive informacije o sastanku te na njima može učiniti izmjene kao promjene naslova ili opisa, vremena odvijanja sastanka te mogućnost dodavanja ili uklanjanja sudionika iz sastanka. U ovom dijelu aplikacije također je uvedena validacija podataka kao i kod stranice za stvaranje sastanka. Izgled stranice za uređivanje korisnika prikazan je na slici 5.5.



Slika 5.5 Stranica za uređivanje sastanaka

Nakon što su prikazane sve funkcionalnosti Blazor dijela aplikacije, moguće je opisati rad cijele aplikacije kao cjeline. Kada korisnik otvori web aplikaciju na njoj može

vidjeti kreirane sastanke te može i sam kreirati poziv za sastanak. Stvaranjem poziva za sastanak, pozvanim sudionicima se odmah šalje poruka e-pošte te sudionici mogu putem nje odbiti ili prihvatiti poziv. Kako izgleda primjer poziva koju sudionici dobiju prikazan je na slici 5.6.



Slika 5.6 Poziv za sastanak

U suštini, opisane su sve mogućnosti aplikacije te sama aplikacija radi kako je planirano no postoje još neke mogućnosti i ideje za poboljšanja. Kao neki budući dodatci aplikaciji mogu se navesti komunikacija pomoću Google servisa kao što su Google Meet i Google kalendar no u trenutku razvoja implementiranje tih Google servisa zahtijevali bi previše vremena. Uz vrijeme za implementiranje željenih servisa potrebno je i znanje u Google Cloud Console platformi. Aplikaciju bi također mogla poboljšati funkcija prijave i registracije korisnika što u ovom trenutku nije u njoj. Problem u razvoju tih funkcija bio je taj što Blazor kao razvojni okvir još nije dovoljno popularizirani te je cijeli projekt bilo potrebno spustiti na nižu verziju .NET-a, a u toj verziji nisu dovoljno dobro određene funkcionalnosti samog razvojnog okvira. Smatram da bi za izradu ovakve web aplikacije bilo bolje koristiti neki drugi razvojni okvir kao Angular na primjer.

6. ZAKLJUČAK

U radu su opisani razvojni okviri na strani poslužitelja i korisnika kao i rad s bazom podataka. Cilj rada bio je upoznati se s .NET razvojnim okvirom kao i s njihovim alternativama.

Kao rezultat nastala je web aplikacija za dogovaranje sastanaka s komunikacijom preko vanjskog servisa. Aplikacija pruža mogućnost dogovaranja sastanaka gdje korisnik stvara događaj te putem e-pošte poziva druge sudionike, a kao povratnu informaciju pozvani sudionici imaju mogućnost potvrditi dolaze li ili ne dolaze na sastanak.

Iako je aplikacija uspješno realizirana još uvijek ima određenih ideja kako ju usavršiti. Neke od ideje kako bi aplikacija bila još bolja su mogućnosti prijave i registracije korisnika te dodavanje mogućnosti korištenja Google Meeta pri samom stvaranju sastanka. U trenutnoj fazi razvoja aplikacija još uvijek radi samo lokalno i razvijena je u .NET verziji 7 koji neće imati dugotrajnu podršku pošto je na tržištu .NET verzija 8. Problem razvoja u starijoj verziji nosi gubitak podrške za određene servise te je samim tim bio problem realizirati određene ciljeve. Vrijedi razmotriti korištenje nekih drugih razvojnih okvira posebice umjesto Blazora koji još nije dovoljno globalno zaživio te mu je potreban dodatan razvoj.

Razvijena aplikacija je dobar temelj za razvoj no potrebno je daljnje usavršavanje kako bi se mogla koristiti interno unutar neke organizacije ili globalno.

7. LITERATURA

[1] Guru99. Django vodič za početnike: značajke, Architekstura i povijest 2024. Dostupno na: https://www.guru99.com/hr/django-tutorial.html (6.9.2024.)

[2] Vrđuka Robert. Razvoj web aplikacija u programskom jeziku Ruby on Rails. 2023. Dostupno na: https://repozitorij.foi.unizg.hr/islandora/object/foi:7902/datastream/PDF/view (11.9.2024.)

[3]Prof. dr. Dražen Drašković. Univerzitet u Beogradu – Elektrotehnički fakultet Spring Boot [Online]. 2024. Dostupno na: https://rti.etf.bg.ac.rs/rti/ir4pia/materijali/predavanja/PIA_SpringBoot.pdf (10.9.2024.)

[4] limun.co. Što je Angular i za što se koristi. Dostupno na: https://limun.co/bh/blog/315-sta-je-angular-i-za-sta-se-koristi (16.9.2024.)

[5] Šta je React? PopArt Studio. Dostupno na: https://www.popwebdesign.net/sta-je-react.html (17.9.2024.)

[6] Programiranje.com.hr 2023. Dostupno na: https://programiranje.com.hr/blazor-osnove (12.9.2024.)

8. OZNAKE I KRATICE

API - Application Programming Interface

CLR - Common Language Runtime

CRUD - Create, Read, Update, Delete

CSS - Cascading Style Sheets

HTML - HyperText Markup Language

ID - Identifier

MVC - Model View Contreller

MVT - Model View Template

URL - Uniform Resource Locator

XSS - Cross-Site Scripting

9. SAŽETAK

Naslov: Web aplikacija za organizaciju sastanaka i događaja

Rad se bavi razvojem web aplikacije za dogovaranje sastanaka uz korištenje vanjskog servisa kao i upoznavanje s radom .NET razvojnih okvira. Aplikacija korisnicima pruža mogućnost stvaranja sastanka i pozivanjem određenog broja sudionika. Svaki sudionik može odabrati hoće li sudjelovati na sastanku ili ne.

Rezultat rada je aplikacija i spoznaje o njenome razvoju, kao i dobar temelj kao ideja za daljnji razvoj i usavršavanje.

Ključne riječi: web aplikacija, .NET, razvojni okviri.

10. ABSTRACT

Title: Web application for meeting and event organization

The paper deals with the development of a web application for scheduling meetings, utilizing an external service, and exploring the workings of .NET development frameworks. The application allows users to create a meeting and invite a specified number of participants. Each participant can choose whether to attend the meeting or not.

The outcome of the paper is both the application itself and the knowledge gained during its development, providing a solid foundation and idea for further development and enhancement.

Keywords: web application, .NET, development frameworks.

IZJAVA O AUTORSTVU ZAVRŠNOG RADA

Pod punom odgovornošću izjavljujem da sam ovaj rad izradio/la samostalno, poštujući načela akademske čestitosti, pravila struke te pravila i norme standardnog hrvatskog jezika. Rad je moje autorsko djelo i svi su preuzeti citati i parafraze u njemu primjereno označeni.

Mjesto i datum	Ime i prezime studenta/ice	Potpis studenta/ice
U Bjelovaru, 22.10.2024.	FILIP MARIĆ	Filip Marić

U skladu s čl. 58, st. 5 Zakona o visokom obrazovanju i znanstvenoj djelatnosti, Veleučilište u Bjelovaru dužno je u roku od 30 dana od dana obrane završnog rada objaviti elektroničke inačice završnih radova studenata Veleučilišta u Bjelovaru u nacionalnom repozitoriju.

Suglasnost za pravo pristupa elektroničkoj inačici završnog rada u nacionalnom repozitoriju

FILIP MARIĆ

ime i prezime studenta/ice

Dajem suglasnost da tekst mojeg završnog rada u repozitorij Nacionalne i sveučilišne knjižnice u Zagrebu bude pohranjen s pravom pristupa (zaokružiti jedno od ponuđenog):

a) Rad javno dostupan
b) Rad javno dostupan nakon _____ (upisati datum)
c) Rad dostupan svim korisnicima iz sustava znanosti i visokog obrazovanja RH
d) Rad dostupan samo korisnicima matične ustanove (Veleučilište u Bjelovaru)
e) Rad nije dostupan

Svojim potpisom potvrđujem istovjetnost tiskane i elektroničke inačice završnog rada.

U Bjelovaru, 22. 10. 2024.

Filip Marić

potpis studenta/ice