Analiza mogućnosti migracije C#.Net stolnih aplikacija na web tehnologije

Šumak, Antonio

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonio Šumak

ANALIZA MOGUĆNOSTI MIGRACIJE C#.NET STOLNIH APLIKACIJA NA WEB TEHNOLOGIJE

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Antonio Šumak

Matični broj: 0016137004

Studij: Poslovni sustavi

ANALIZA MOGUĆNOSTI MIGRACIJE C#.NET STOLNIH APLIKACIJA NA WEB TEHNOLOGIJE

ZAVRŠNI/DIPLOMSKI RAD

Mentor:

Izv. prof. dr. sc. Zlatko Stapić

Varaždin, rujan 2023.

Antonio Šumak

Izjava o izvornosti

Izjavljujem da je moj završni rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U radu opisane su tehnologije koje služe za razvoj stolnih i web aplikacija, ali isto tako opisano je sve bitno vezano uz migracije. Uvodni dio opisuje tehnologije koje se koriste za izradu korisničkih sučelja, serverskih aplikacija i stolnih aplikacija. Svaka od te tehnologije dodatno je opisana te su navedene prednosti i nedostaci svih tehnologija. Na kraju poglavlja prikazane su sintakse odabranih tehnologija. Zatim u sljedećem poglavlju su opisane migracije te zašto uopće dolazi do potrebe za migriranjem aplikacije. Nakon globalnog uvoda u migracije dodatno su opisane tehnike i najbolje prakse migriranja. Praktičan dio opisan je kroz nekoliko poglavalja u kojima se opisuje postupak migriranja stolne aplikacije na web.

Ključne riječi: migracija, C#, .NET framework, React, web, stolna aplikacija, razvoj programskog proizvoda

Sadržaj

1. Uvod

U zadnjih nekoliko godina došlo je do velikog napretka tehnologija. Taj napredak najviše se osjetio u svijetu web tehnologija. Web je postao jako popularan i sve više aplikacija biva smješteno na web mjesta. Upravo ta popularnost interneta i Internet tehnologija razlog je pisanja ovog završnog rada. Cilj završnog rada je istražiti te teorijski opisati mogućnosti migracije postojećih programskih proizvoda pisanih u tehnologiji C#.NET na web okruženja. Prednost pokretanja aplikacija na web mjestu je kompatibilnost koju web pruža, u smislu da neovisno o uređaju aplikacija će raditi. Do nedavno prednost stolnih aplikacija nad web aplikacijama bila je način rada bez povezanosti na mrežu, no pojavom PWA (Progressive Web App) tehnologije taj način rada sada je dostupan i za web aplikacije.

Kako bi stolna aplikacija proradila na web-u potrebno je odraditi migraciju. Te preinake možemo kategorizirati pod migraciju korisničkog sučelja, migraciju poslovne logike i migraciju rada sa podacima. Stolna aplikacija koja će biti pretvorena u web aplikaciju pisana je u .NET Framework-u 4.8, a baza koja se koristi je SQLlite baza podataka. Kroz istraživanje potrebno je utvrditi koja će se tehnologija pokazati kao najbolja opcija za migraciju postojeće aplikacije. Kako se radi o aplikaciji za stolna računala korisničko sučelje će sigurno biti migrirano u jednu od web tehnologija. Tehnologija u koju će biti migrirano korisničko sučelje biti će određena usporedbom modernih tehnologija za kreiranje web sučelja te odabirom one koja se pokaže kao najpogodnije rješenje.

U radu će prvo biti opisane tehnologije koje su bile na odabir te prednosti i nedostaci istih. Primjeri aplikacije kreirani u tim tehnologijama kako bi se tehnologije lakše usporedile. Nakon tehnologija biti će opisane migracije, što su migracije kao pojam i koji su svi potrebni koraci kako bi se sama migracija uspješno provela i izvršila. Na kraju biti će prikazano i objašnjeno kako je sve odrađeno i šta je moglo bit bolje i načini kako to poboljšati.

2. Metode i tehnike rada

Prilikom obrade teme rada najviše su korištene knjige, ali isto tako i izvori pisani od strane različitih tvrtki. Te izvore možemo svrstati kao opis usluge koju tvrtka pruža, blogovi tvrtki pisani od strana zaposlenika. Uz navedene izvore korištene su i dokumentacije odabranih tehnologija i programskih jezika.

Ideja praktičnog dijela rada je dokazivanje mogućnosti migracije stolne .NET aplikacije (Windows Forms biblioteke) na web. Kako je riječ o migraciji potrebno je iskoristiti što više postojećeg koda. Poslužiteljski dio pisan je u C#-u (.NET Framework) te je namjera da se poslovna logika i način rada s podacima zadrži na način kakav je, dok će se klijentski dio (Windows forms dio) iznova razviti u nekoj od web tehnologija za izradu korisničkih sučelja. Web aplikacija i stolna aplikacija moraju dijeliti istu bazu podataka kako bi poslovna logika i rad s podacima ostao isti.

Kako bi provedba praktičnog dijela bila moguća potrebno je na računalu imati sljedeće alate i tehnologije:

- Visual Studio Community 2022 – besplatan alat za razvoj .NET aplikacija
- Visual Studio Code – besplatan alat za razvoj korisničkih sučelja
- .NET Framework 4.7.2 SDK – razvojni paket u kojem je razvijena i stolna i web aplikacija
- NodeJs – tehnologija koja je potrebna kako bi se izradilo korisničko sučelje
- DBeaver Community – program za upravljanje i pregled baze podataka

3. Opis tehnologija

Purewal [1] govori kako se web aplikacije sastoje od 3 glavna dijela. Klijentski dio koji predstavlja dio aplikacije koju koristi korisnik, serverski dio u kojem se vrši obrada podataka i slanje tih podataka na klijenta te serverski dio na kojem je smještena baza podataka. Uz sve nabrojeno potrebno nam je i web mjesto preko kojeg ćemo pristupiti našoj web stranici/aplikaciji. Pri kreiranju aplikacije sučelja Powell i Mikowski [2] navode kako su SPA (jednostrane aplikacije) normalan način kreiranja sučelja u proteklih nekoliko godina. „SPA aplikacija je aplikacija koja se isporučuje u pregledniku na način da se ne učitava ponovno prilikom korištenja“ [2, p. 4] Fender i Young [3] u svojoj knjizi govore kako JavaScript tehnologije izmiču kontroli, to jest kako je tih tehnologija sve više i više te kako korisnik ima veliku količinu tehnologija na raspolaganje. Bez obzira što su te tehnologije pokrenute JavaScript-om ne znači da su iste. Od velikog broja tehnologija moguće je vidjeti kako se razlikuju u velikom broju značajki. Codemotion Magazine [4] pokazuje kako je izbor između tehnologija velik te kako je potrebno odabrati tehnologiju prema svojim potrebama. Svaka tehnologija bolja je u nekim stvarima no ne postoji savršena tehnologija koja bi sve te značajke spojila u jednu tehnologiju. Iz tog razloga potrebno se prilagoditi potrebama.

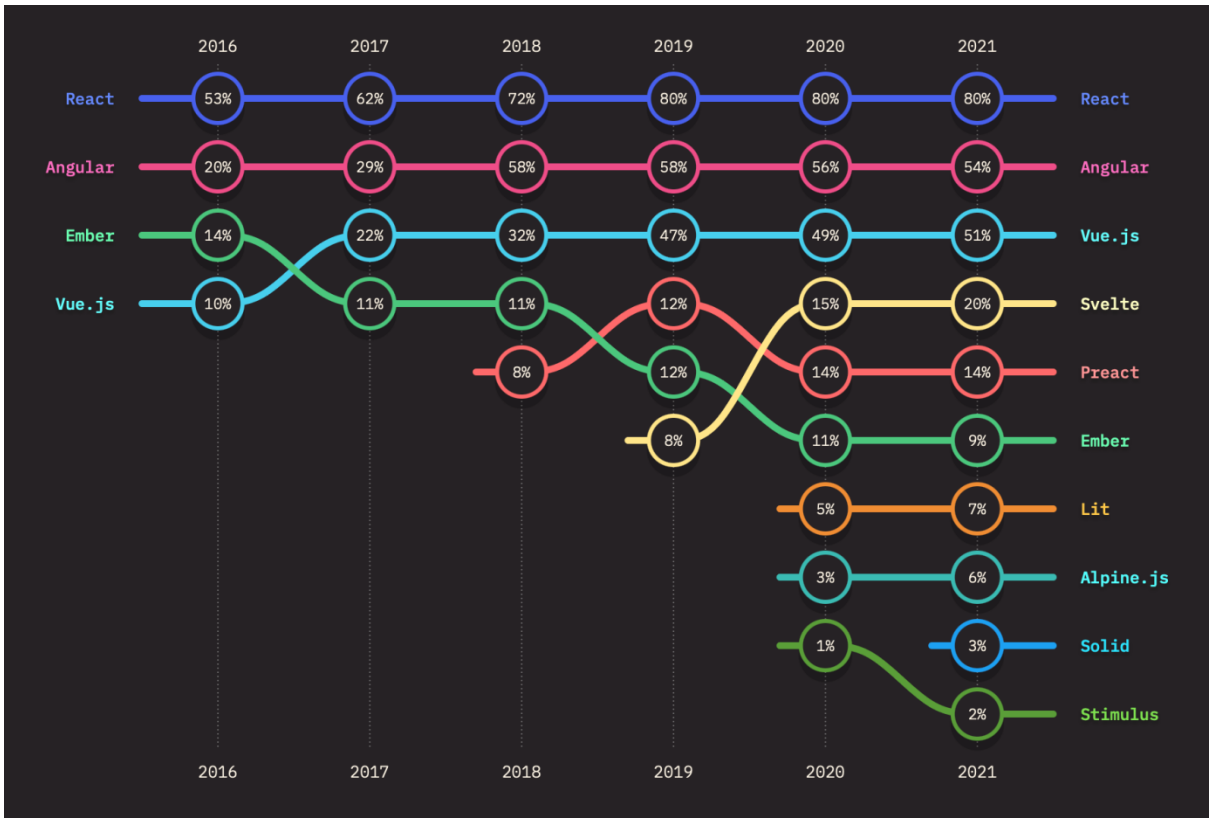
Kako je spomenuto već od strane Purewal-a uz tehnologiju za kreiranje sučelja potrebna nam je tehnologija koja će uzimati podatke iz baze podataka te podatke obraditi i poslati na korisničku aplikaciju. Clark [5] piše kako ni odabir programskog jezika za server nije jednostavan. Iz priloženog članka vidljivo je kako postoji velik broj jezika koji još dodatno imaju poveći broj tehnologija. Svaki jezik ima svoje dodatne značajke no u globalu svi rade istu stvar, ali na različite načine. Kako je već spomenuto cilj tehnologija na serveru je spremanje i izvlačenje podataka iz i u bazu podataka. Zatim se ti podaci formatiraju te ako postoji dodatno se izvršava poslovna logika i obrađeni podaci se šalju na sučelje. Masse [6] ovakav tip komunikacije naziva komunikacija putem REST API servisa. Važno je naglasiti kako je Masse također naglasio kako je ovo dvosmjerna komunikacija, to jest komunikacija u kojoj protok podataka ide od klijenta na server i sa servera na klijenta.

Kroz sljedeća tri poglavlja proći će se kroz neke od tehnologija za izradu korisničkih sučelja (eng. Front-end), tehnologija za izradu aplikacija na serveru i tehnologija za izradu aplikacija za stolna računala. Usporedbom tih tehnologija odabrati će se tehnologija koja je po karakteristikama bolja od drugih.

3.1. Tehnologije za izradu korisničkih sučelja

Svaka tehnologija posebna je na svoj način, ali na kraju sve tehnologije imaju isti cilj, a to je kreiranje jednostavnih, održivih, skalabilnih komponenta koje je moguće iskoristi više puta.

Prema State of JS [7] trenutno najkorištenija tehnologija za izradu korisničkih sučelja je React. React već dugi niz godina drži vodeće mjesto u tome koliko se koristi. Iz grafikona se može se iščitati kako je React od početka istraživanja vodeća tehnologija.



Slika 1. Poredak popularnosti front-end tehnologija [7]

3.1.1. React

React je JavaScript tehnologija za kreiranje korisničkih sučelja. Stvoren je 2013. godine od strane Facebook-a. Prema React dokumentaciji [8] trenutno najnovija verzija React tehnologije je 18.2.0. Kroz godine tehnologija je prošla kroz 25 verzija. Najznačajnija nadogradnja pojavila se u verziji 16.8. kad se iz klasnog načina stvaranja komponenti prešlo na funkcijski način pisanja komponenti. Ta značajka olakšala je i ubrzala razvoj. Ekosustav Reacta je iznimno velik jer je React kao tehnologija zamišljen da se oslanja na vanjske tehnologije. Sustav za upravljanje stanjem aplikacije, promjena ruta unutar lokacije, upravljanje

formama samo su neki od primjera [9]. React osim što je sam po sebi tehnologija služi kao podloga za druge tehnologije kao što su NextJs, Remix, Gatsby. Prethodno navedene tehnologije samo su od nekih koje postoje, ali te su najkorištenije od svih.

3.1.1.1. Prednosti

U poglavlju prednosti proći će se kroz neke od najznačajnijih prednosti tehnologije. Prisutnost Virtualnog DOMa, velika podrška zajednice, jednostavnost učenja o tehnologiji, podrška za razvoj mobilnih aplikacija pomoću React Native tehnologije te velika traženost tehnologije samo su od nekih prednosti tehnologije.

- Virtualni DOM – Zahvaljujući virtualnom DOM-u React ažurira komponente u kojima se dogodila promjena, a ne svim komponentama koje su u aplikaciji. Na taj način se minimizira broj interakcija sa stvarnim DOM-om [10].

- Podrška zajednice – Jedna od jačih prednosti React-a je njegova velika zajednica. Postoji velika količina proizvođača gotovih komponenti (MUI, Ant Design, Semantic UI, Chakra UI, Theme UI…) [11]. Isto kao i za komponente tako i za sva pitanja vezana uz React negdje na internetu postoji odgovor na to pitanje [10].

- Jednostavnost učenja – React za razliku od ostalih tehnologija nema nikakvih složenih koncepata za savladati. Sastoji se od pisanja HTML-a i JavaScript-a. Potrebno je malo vremena da se savlada JSX sintaksa, ali s vremenom se navikne [10].

- React native – React native je tehnologija za kreiranje mobilnih aplikacija pomoću React-a. Vrlo slično web verziji tehnologije. Iz tog razloga tranzicija iz web razvoja u mobilni razvoj nije problematična [12].

- Traženost na tržištu – Zbog ogromne popularnosti među zajednicom React je najtraženiji i najbolje plaćen među tehnologijama za izradu korisničkih sučelja [7].

3.1.1.2. Nedostaci

Nedostaci tehnologije su neizbježna stvar. Svaka tehnologija ima neke nedostatke, neke tehnologije imaju više nedostataka dok druge manje. Začetnici tehnologije React

nedostatke su sveli na minimum. Najznačajniji nedostaci su loša dokumentacija i JSX sintaksa koju React zahtjeva.

- Loša dokumentacija – Dokumentacija u trenutnoj izvedbi zbog zastarjelosti nije prigodna za početnike. Isto tako sam protok procesa učenja nije najbolje posložen. Ima podosta skakanja između tema. Još jedan od razloga zašto je dokumentacija lošija je prebrz razvoj tehnologije.

- JSX – JSX je kombinacija HTML-a i JavaScript-a [8]. Zbog miješanja tih dviju stvari dolazi do manje čitkosti koda. JSX ima i sitan utjecaj na performanse React-a iz razloga jer se taj JSX dodatno mora pretvarati u HTML i JavaScript [10].

3.1.2. Angular

Angular je JavaScript tehnologija za kreiranje korisničkih sučelja. Kreiran je 2009. godine od strane Google-a. Najviše se koristi za kreiranje srednjih do velikih korisničkih sučelja. Za razliku od ostalih tehnologija primaran jezik nije JavaScript nego TypeScript. Na taj način ostvaruje se sigurnost [13]. Isto kao i React Angular je imao veliku promjenu u pisanju koda. To se dogodilo kad se prelazilo sa AngularJS-a na Angular 2. To jest prijelaz sa JavaScript verzije na TypeScript verziju tehnologije. Također za razliku od većine tehnologija Angular već u sebi ima ugrađene sve potrebne alate [13].

3.1.2.1. Prednosti

Isto kao i React i Angular ima svoje prednosti. Te prednosti su donekle drugačije u odnosu na prednosti React tehnologije, što pokazuje zašto bi netko odabrao Angular prije nego React. Neke od značajnijih prednosti su Angularov zatvoreni sustav, TypeScript integracija, podrška zajednice te dvosmjerni protok podataka.

- Zatvoren sustav – Angular je zatvoren sustav u smislu da sve potrebne biblioteke dolaze instalirane sa instalacijom Angulara. Kako dolaze sa instalacijom osigurano je da biblioteke najbolje rade sa tehnologijom. Ne postoji oslanjanje na vanjske biblioteke kao kod nekih tehnologija [13].

- TypeScript – TypeScript je u odnosu na JavaScript nadogradnja. Sintaksa pisanja koda ostaje ista, ali TypeScript je programski jezik koji ima tipove i jezik koji upozorava na

greške prilikom pisanja koda, a ne tek dok je aplikacija u fazi izvršavanja [14]. Korištenje TypeScript-a kao primarnog jezika omogućava bolje iskustvo pisanja koda, ali isto tako i povećanu sigurnosti prilikom pisanja koda.

- Podrška zajednice – Isto kao i kod React-a postoji velika zajednica koja olakšava život programerima sa već gotovim komponentama i odgovorima na veliku većinu pitanja.

- Dvosmjerni protok podataka – Prema google dokumentaciji ako dođe do promjene vrijednosti dvosmjerni protok podataka omogućava promjenu vrijednosti na svim mjestima gdje se ta vrijednost koristi [13].

3.1.2.2. Nedostaci

U odnosu na React i Angular ima nekoliko nedostataka zbog kojih tehnologija nije toliko voljena te zbog čega tehnologiji opada popularnost u svijetu JavaScript tehnologija za izradu korisničkih sučelja. Ti nedostaci su strma krivulja učenja, kompleksnost alata koji dolaze sa Angularom pri instalaciji i opadanje populanosti.

- Strma krivulja učenja – Zbog svog tog zatvorenog sustava i koncepata na kojima se Angular temelji dosta je teško pohvatati sve te koncepte. Zato je potrebno vrijeme da se sve to savlada [12].

- Kompleksnost alata – Isto kao i Angular alati koji dolaze instalirani zajedno sa tehnologijom imaju strmu krivulju učenja. Zbog koncepata na kojima se Angular temelji potrebno je neko vrijeme da se zapamti potrebna sintaksa za neke od alata [12].

- Opadanje popularnosti – Na slici 1. moguće je vidjeti kako popularnost opada. Razlog toga je dolazak novih tehnologija koje su manje i brže, ali i puno jednostavnije nego Angular [7].

3.1.3. Vue

Vue je JavaScript tehnologija za kreiranje korisničkih sučelja. Tehnologija je kreirana 2014. godine od strane Evan You-a. Za razliku od Reacta i Angulara, Vue je kreiran od strane jedne osobe te iz tog razloga nema toliku popularnost kao prethodne dvije tehnologije. Isto kao i Angular i Vue prošao je kroz veliku promjenu. Trenutna verzija Vue-a, verzija 3 prepisana je na novo iz JavaScripta u TypeScript kako bi i u Vue omogućila podrška za TypeScript [15]. Za

razliku od Angulara koji može biti pisan na samo jedan način, React koji može biti pisan na 2 načina (funkcijsko i klasno) Vue može biti pisan na 3 načina (Options, Composition i Class API).

3.1.3.1. Prednosti

Prednosti tehnologije Vue slične su i Reactu i Angularu, ali postoje prednosti koje su specifične samo za Vue. Prema ovim prednostima moglo bi se zaključiti kako je Vue negdje između Reacta i Angulara što je u praksi istina. Neke od tih prednosti su jednostavnost učenja, dvosmjerni protok podataka te brzina i veličina.

- Jednostavnost učenja – Vue komponente sastoje se od 3 dijela. Prvi dio je „template“ dio koji je zapravo HTML sa Vue dodacima. Ti dodaci su direktive koje izgledaju kao parametri za HTML element. Zatim imamo JavaScript/TypeScript „script“ dio u kojem se na jedan od 3 načina piše logika vezana uz komponentu. I na kraju zadnji dio komponente je „style“ to jest CSS dio. Vrlo slično standardnom pristupu pisanja sa HTML-om i JavaScript-om. Samo što ovdje script dio ima elemente koji su vezani uz Vue [15].

- Dvosmjerni protok podataka – Unutar Vue-a taj se dvosmjerni protok podataka najčešće koristi pri elementima za forme. Vrlo jednostavno povezivanje događaja i vrijednosti uz pomoć direktive „v-model“ [15]. Značenje dvosmjernog protoka podataka je isto kao i kod Angulara.

- Brzina i veličina – Vue je kao tehnologija vrlo malen. Svega 18kB. Takvo svojstvo također pozitivno utječe i na brzinu aplikacije. Kako tehnologija nije napuhnuta nepotrebnim dodacima sam proces razvoja je brži [16].

3.1.3.2. Nedostaci

Za razliku od prethodne dvije navedene tehnologije Vue ima najkritičnije nedostatke. Iz tog razloga moglo bi se zaključiti kako je Vue i najmanje korištena tehnologija između ove tri nabrojane tehnologije što je u praksi istinito. Nedostaci React-a i Angular-a većinom su problemi prilagodbe (npr. kompleksnost i sintaksa). Dok nedostaci Vue tehnologije povezani su sa ograničenjima tehnologije.

Nedostatak dodataka od zajednice – Ako se usporedi broj komponenti i biblioteka koje su dostupne za React ili Angular vidi se kako je velika razlika u broju dostupnih komponenti za Vue. To je plod toga što Vue nije jako popularna tehnologija [16].

Nije preporučen za velike projekte – Vue je preporučen za manje i srednje projekte. Razlog toga je nezrelost tehnologije, ali isto tako i nedostatak financijske podrške od velikih kompanija.

Fleksibilnost – Vue komponentu moguće je napisati na 3 različita načina. Ova fleksibilnost zvuči odlično na prvu, ali kad se korisnicima pruži tolika fleksibilnost obično dolazi do komplikacija jer nekima više odgovara jedan način, a drugima drugi način [16].

3.1.4. Prikaz jednostavne aplikacije

Uz sve navedene prednosti i nedostatke koji su bili navedeni u prethodnom poglavlju bitno je vidjeti kako izgleda sintaksa tehnologije. Sintaksa i količina koda koju je potrebno pisati iste je važnosti kao i prednosti i nedostaci tehnologije. U sljedećim isječcima kodova prikazano je kako kod iste aplikacija izgleda u različitim tehnologijama. Na sljedećoj slici (Slika 2.) prikazan je izgled aplikacije koju je bilo potrebno rekreirati odabranim tehnologijama.

Slika 2. Prikaz jednostavne aplikacije [autorski rad]

3.1.4.1. React

Isječak koda 1 pokazuje kako bi izgledala jednostavna aplikacija u React-u koja osvježi brojač na pritisak gumba i prikaže ga na zaslonu. Postoji samo jedna datoteka, takozvana komponenta unutar jedne datoteke (eng. Single file component) te je također moguće vidjeti kako je komponenta obična funkcija. Imamo jedanu varijablu „counter“ koja predstavlja stanje brojača koja se zatim unutar funkcije „increaseCounter“ klikom na gumb povećava za 1. Na slici je također moguće vidjeti JSX sintaksu komponente. Iz isječka koda 1. moguće je vidjeti kako je JSX sintaksa samo kombinacija običnog HTMLa i JavaScripta.

```
import * as React from 'react';
import './style.css';

export default function App() {
  const [counter, setCounter] = React.useState<number>(0);
  const increaseCounter = () => {
    setCounter(counter + 1);
  };
  return (
    <section className="main-section">
      <div className="inner-wrapper">
        <div>
          <h2>Counter value</h2>
          <p className="counter">{counter}</p>
        </div>
        <button onClick={increaseCounter}>Increase counter</button>
      </div>
    </section>
  );
}
```

Isječak koda 1. Prikaz aplikacije brojača React [autorski rad]

3.1.4.2. Angular

Za razliku od Reacta za Angular su nam potrebne minimalno dvije datoteke kako je moguće vidjeti na isječku koda 2. Jedna HTML i jedna TypeScript datoteka. HTML dio je običan HTML koji sadrži Angular svojstva. Dok TypeScript datoteka je klasa, a ne funkcija kao što je to kod React-a.

app.component.html

```
<section class="main-section">
    <div class="inner-wrapper">
      <div>
        <h2>Counter</h2>
        <p class="counter">{{ counter }}</p>
      </div>
      <button (click)="incrementCounter()">increment</button>
    </div>
  </section>
```

app.component.ts

```
import { Component } from '@angular/core';

@Component({
  selector: 'my-app',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
})
export class AppComponent {
  constructor() {}

  counter = 0;

  incrementCounter() {
    this.counter++;
  }
}
```

Isječak koda 2. Prikaz aplikacije brojača Angular [autorski rad]

3.1.4.3. Vue

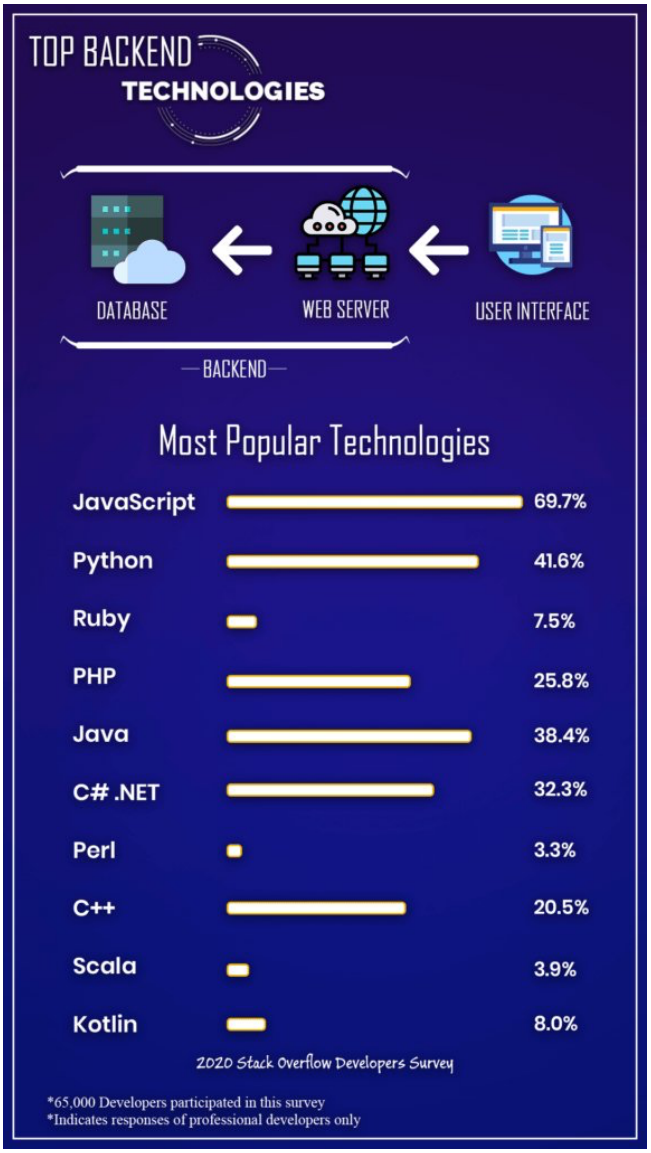
Na isječku koda 3 prikazana je Vue komponenta gdje se opet vraća na pojam komponente unutar jedne datoteke. HTML je normalan HTML, ali sa nekim Vue dodacima (npr. @click događaj). Dok „script“ blok je jedan veliki objekt koji ima predefinirane ključeve. Na slici možemo vidjeti „data“ funkciju. Unutar navedene funkcije imamo vrijednosti koje predstavljaju stanje komponente. Zatim imamo objekt „methods“ u kojeg pišemo metode koje koristimo unutar komponente. Ovaj način pisanja komponente naziva se Options API.

```
<template>
  <section class="main-section">
    <div class="inner-wrapper">
      <div>
        <h2>Counter</h2>
        <p class="counter">{{ counter }}</p>
      </div>
      <button @click="increaseCounter">Increase
counter</button>
    </div>
  </section>
</template>

<script>
export default {
  name: 'App',
  data() {
    return {
      counter: 0,
    };
  },
  methods: {
    increaseCounter() {
      this.counter++;
    },
  },
};
</script>
```

Isječak koda 3. Prikaz aplikacije brojača Vue [autorski rad]

3.2. Tehnologije za izradu serverskih aplikacija

Isto kao i kod tehnologija za izradu sučelja tako i za tehnologije za server vrijedi da postoji velik izbor tehnologija. No za razliku od tehnologija za izradu korisničkih sučelja, za serverske aplikacije postoji puno veći broj programskih jezika između kojih možemo birati. Prilikom pisanja koda za server nema toliko restrikcija kao kod kreiranja korisničkih sučelja. Velika većina programskih jezika može izvršiti neku logiku. Kod pisanja koda za server bitno je unaprijed pokušati odrediti koliko će ljudi koristiti aplikaciju, kako bi mogli odrediti potrebe kao što su skaliranje servera i odabir pravih paketa za uslugu posluživanja [5].



Slika 3. Prikaz najpopularnijih tehnologija [5]

3.2.1. JavaScript

Prema slici 5. JavaScript je najpopularniji jezik za izgradnju serverskih web aplikacija. JavaScript je programski jezik koji se izvodi na strani klijenta. To znači da JavaScript nije stvoren kao serverski jezik. Ali zahvaljujući NodeJs JavaScript kod moguće je pokrenuti na serveru kao serversku aplikaciju. NodeJs je jednojezgreni asinkroni programski jezik za izgradnju skalabilnih web aplikacija [17].

3.2.1.1. Prednosti

JavaScript jako je poznat jezik te iz tog razloga mora postojati neki razlog zašto ga programeri koriste u svojim projektima. JavaScript ima velik prednosti u odnosu na druge programske jezike. Neke od značajnijih prednosti su brzina razvoja koju JavaScript omogućuje, minimalnost jezika, zbog jednostavnosti minimizira troškove te zbog velike popularnosti ima veliku zajednicu.

- Brzina razvoja – JavaScript je vrlo jednostavan i lagan (u smislu performansa) jezik kojim se veoma brzo mogu postignuti određene stvari bez previše konfiguriranja. Najbolji primjer je podizanje REST API servera. Kako bi se to postiglo potrebno je 10-ak linija koda, što pokazuje jednostavnost [5] [17].

- Minimalnost jezika - Minimalnost se dobro nadovezuje na brzinu razvoja. JavaScript je vrlo moćan jezik, ali njegova sintaksa omogućava jednostavno pisanje koda, ali isto tako potrebno je manje koda u odnosu na druge jezike kako bi se postigao željeni cilj [5].

- Minimizira troškove – Nadovezuje se na obje prethodne stavke. Ako je brzina razvoja velika i nije potrebno pisati puno početnog koda onda su i troškovi razvoja manji, jer se troši manje vremena na razvoj.

- Zajednica – JavaScript zajednica programera je jedna od većih zajednica na internetu. Samo ako se pogleda broj npm paketa može se vidjeti sama popularnost ovog jezika.

3.2.1.2. Nedostaci

JavaScript je kroz godine postojanja postao zreliji jezik. Usporedivši situaciju jezika pri izlasku napredak je ogroman. Nedostaci jezika u trenutnoj fazi postojanja su brzina, to jest nedostatak brzine, nedostatak dinamički pisanog jezika te problematična integracija sa MYSQL bazom podataka. Neke od nedostataka nije moguće izbjeći zbog ograničenja od strane V8 jezgre preglednika.

- Brzina – JavaScript na raspolaganju ima samo jednu jezgru, što znači da ne postoji mogućnost višedretvenosti. Postoji rješenje za taj problem tzv. Web Workers, ali

dodatno kompliciraju stvar. To je jedan od razloga zašto je izvršavanje JavaScript koda sporo u odnosu na ostale programske jezike kao što su Go, Rust i C# [18].

- Dinamički pisan jezik – JavaScript je tzv. „Dynamically typed language“ što u prijevodu znači da ne zahtjeva specificiranje tipova varijabli. Što je jako loše jer u velikim projektima može doći do zbunjenosti vezane uz pojavu tipova gdje ti tipovi ne bi smjeli bit. Te probleme možemo riješiti na 2 načina. Prvi je da ručno provjeravamo tipove sa „typeof“, ali na taj način zagađujemo kod. Drugi način je da koristimo TypeScript koji ima tipove [18].

- Problematično implementiranje MYSQL baze – JavaScript i MYSQL baza podataka nisu stvoreni jedno za drugo. Moguće je povezati NodeJs i MYSQL bazu, ali je puno jednostavnije napraviti povezivanje sa PostgreSQL. Također alati za NodeJs koji služe kao ORM nemaju punu podršku kao što imaju sa PostgreSQL bazom [5].

3.2.2. Python

Python je trenutno uz JavaScript najpopularniji programski jezik. Iz istih razloga kao što i JavaScript, Python ima širok krug primjene. Koristi se u izradi servera, embedded razvoju (MicroPython), obrada podataka, analiza podataka, izrada GUI stolnih aplikacija, web razvoj samo su od nekih područja.

3.2.2.1. Prednosti

Isto kao i JavaScript Python je jako popularan jezik. Njegova popularnost ukazuje na to da ima prednosti koje neki drugi programski jezici nemaju. Kroz godine razvoja jezik je bio ubrzan, obogaćen dodatnim funkcionalnostima te mu se čak i unaprijedila čitkost koda. Neke od značajnijih prednosti Pythona su jednostavnost, broj biblioteka koje pruža te omogućen razvoj aplikacija niske razine.

- Jednostavnost – Python je jako popularan među početnicima iz razloga jer je napravljen da bude jednostavan za čitanje i pisanje. Postoji velik broj već ugrađenih funkcija koje obavljaju neke zadatke za koje bi bilo potrebno pisati veću količinu koda. Npr. Funkcija „sum“ koja prolazi kroz polje i zbraja sadržaj polja [5].

- Velik broj biblioteka – Zahvaljujući bibliotekama python obuhvaća velik broj područja kao što su obrada i analiza podataka (Pandas), AI i MachineLearning (PyTorch, TensorFlow), izrada igrica (Pygame) samo su od nekih. Razvoj aplikacija u pythonu pojednostavljen je jer se kompleksne stvari rješavaju pomoću biblioteka [19].

- Mogućnost embedded razvoja – Bez obzira što je Python programski jezik visoke razine, postoji mogućnost da se koristi na najnižim razinama. To je moguće zahvaljujući MicroPythonu. Tako se Python može koristiti za programiranje bilo kojeg mikro kontrolera [5].

3.2.2.2. Nedostaci

Kako je već spomenuto kroz godine je Python postajao sve bolji i bolje, no neki nedostaci još su uvijek aktualni. Neki problemi neće moći biti riješeni te će uvijek biti dostupni u programskom jeziku. Nedostaci Pythona su njegov nedostatak brzine, njegova ograničenost bez korištenja vanjskih biblioteka te mogućnost pisanja koda bez tipova.

- Brzina – Zbog opsežnosti i mogućnosti koje Python nudi i vremena potrebnog da se svi te metode visoke razine pretvore u kod niske razine potrebna je određena količina vremena. Iz tog razloga je Python spor [5].

- Ograničenost - Za velik broj mogućnosti potrebno je koristiti dodatne bibliioteke. Za razliku od npr. JavaScripta koji velik broj funkcionalnosti ima uključen bez ikakvih biblioteka [5].

- Mogućnost pisanja koda bez tipova – Za razliku od JavaScripta python ima mogućnost bit pisan sa tipovima podataka ili bez, ako se piše bez tipova podataka onda se moraju raditi dodatne provjere tipova kao i kod JavaScripta.

3.2.3. C#

C# je programski jezik kreiran od strane Microsofta. Jezik ima jako široku primjenu. Koristi se kao programski jezik pisanja skripti za Unity game engine. Također se koristi za kreiranje backend aplikacija, razvoj stolnih aplikacija, ali postoje i tehnologije koje koriste C#

za frontend razvoj u kombinaciji sa HTML-om (Blazor). Svoju primjenu nedavno našao je i u ML-u (ML.NET). Sintaksom je dosta sličan Javi, to jest Java je vrlo vjerojatno imala utjecaj na sintaksu jer je jezik kreiran 5 godina nakon objave Jave [5] [20].

3.2.3.1. Prednosti

C# je najnoviji programski jezik od prethodna dva jezika. Po jeziku je vidljivo kako je jezik nastao utjecajem više programskih jezika (C++ i Java). Taj utjecaj je ostavio pozitivan dojam na funkcionalnost jezika. Neke od prednosti C# jezika su to što je striktno objektno orijentiran, kompatibilan je sa prethodnim verzijama jezika, moguće ga je pisati na svim operacijskim sustavima i postoji mogućnost pristupa memoriji.

- Striktno objektno orijentiran – C# je objektno orijentiran jezik što znači da je kod strukturiran u klase što developeru kasnije olakšava razvoj aplikacija [5].

- Kompatibilnost – Kod koji se koristi na različitim verzijama jezika je kompatibilan ili sa novijim ili sa starijim verzijama jezika. Što je jako korisno ako postoji neka starija aplikacija koja je velika, vrlo lagano se takav tip aplikacije može prebaciti na najnoviju verziju [5] [20].

- Cross-platform – Pisanje C# jezika moguće je na svim operacijskim sustavima koristeći .NET [5].

- Pristup memoriji – Bez obzira što je C# smatran programskim jezikom više razine postoji mogućnost kreiranja pokazivača i samim time postoji mogućnost pristupanja memoriji [5].

3.2.3.2. Nedostaci

Jezik je od samog početka imao jako dobru bazu na temelju koje je bio kreiran. Također kroz godine razvoja velik broj nedostataka bio je popravljen. Trenutno C# je jako stabilan, moćan i brz programski jezik. No neki od nedostataka su sintaksa i zbunjujući nazivi tehnologija kojima su nazvane verzije.

- Sintaksa – Poprilična količina teksta i „boilerplate“ koda.

- Zbunjujući nazivi – Postoji više naziva i više tehnologija kao npr. .NET, .NET framework, .NET Core itd. Teško je pratiti koji je koji jer svi imaju slične nazive.

3.3. Stolne tehnologije

Nakon web tehnologija prelazimo na stolne tehnologije. Stolne aplikacije se u današnje vrijeme sve manje koriste iz razloga što je došlo do pojave web tehnologija. U današnje vrijeme puno je jeftinije razviti web aplikaciju nego stolnu aplikaciju iz razloga što je web zajednica veća i postoji veći izbor tehnologija ali isto velik razlog je kompatibilnost. Web aplikacija može se pokrenuti na bilo kojem operacijskom sustavu jer ta aplikacija ovisi od preglednika. Dok stolne aplikacije ovise od operacijskog sustava. Microsoft ima odlične alate kao što su WindowsForms i WPF za izradu stolnih aplikacija, ali veliki nedostatak je to što su to aplikacije samo za Windows uređaje [20]. Ako koristimo JavaScript i Electron onda imamo mogućnost razvoja aplikacija za sve operacijske sustave, ali ta Elektron aplikacije je na neki način samo web aplikacija u OS prozoru [21].

3.3.1. Elektron

Elektron je JavaScript tehnologija kreirana od strane Githuba za izradu stolnih aplikacija. Za markup jezike koriste se HTML i CSS dok se kao programski jezik koristi JavaScript. Iz ovog je lagano zaključiti kako je ovo jako slično programiranju web aplikacija [21].

3.3.1.1. Prednosti

Elektron je relativno nova tehnologija koja je preuzela tržište desktop aplikacija. Razlog preuzimanja tržišta su prednosti koje nam ta tehnologija omogućuje. Te prednosti su jednostavnost koja omogućuje pisanje stolnih aplikacija web tehnologijama i kompatibilnost.

- Jednostavnost – Za razvoj koriste se HTML, CSS i JavaScript. To su tehnologije koje se koriste za razvoj web aplikacija te to omogućuje bilo kojem web programeru da bez problema počinje sa razvojem stolnih aplikacija. Također ako osoba nije web programer te tehnologije nisu veoma zahtjevne i teške za naučit u odnosu na C# [22].

- Kompatibilnost – Aplikacije razvijene u elektronu mogu se koristiti na svim stolnim platformama (Windows, Linux, MacOS) [21].

3.3.1.2. Nedostaci

Iz razloga što je tehnologija relativno nova na tržištu postoje nedostaci. Ti nedostaci su vezani uz resurse koje aplikacija koristi. Primarno zbog sustava koji pokreće elektron aplikacije nije moguće poboljšati te nedostatke sve dok proizvođač sustava ne unaprijedi taj sustav. Nedostaci elektrona su veličina finalne aplikacije i prekomjerno iskorištavanje resursa.

- Veličina – Kako je elektron baziran na Chromium pregledniku. Chromium sam po sebi zauzima značajnu količinu memorije zbog veličine i milijuna linija koda. Tako da za neku najjednostavniju aplikaciju moguće je očekivati minimalno aplikaciju veličina 100MB [22].

- Iskorištavanje resursa – Zbog toga što je Elektron napravljen za sve operacijske sustave postoje problemi sa povećanom konzumacijom resursa koje ima na raspolaganje. Naravno da aplikacije koje su striktno pisane za jedan operacijski sustav imaju bolju optimizaciju potrošnje, ali nemaju mogućnost pokretanja na drugim operacijskim sustavima [22].

3.3.2. WindowsForms

Windows forme su UI tehnologija za kreiranje stolnih aplikacija namijenjenih Windows računalima. Windows forme omogućuju rad sa već ugrađenim elementima kao što su slike, combobox komponenta, tablica, gumbi, labele i mnogi drugi elementi [23]. Tim elementima je moguće mijenjat poziciju na ekranu, veličinu, boju itd. Svo to pomicanje moguće je grafičkim putem, dok se podaci i upravljanje sa komponentom povezuju kroz C# programski kod.

3.3.2.1. Prednosti

Prednosti koje Windows forme imaju nad svojom konkurencijom su njihova jednostavnost i brzina razvoja koju pružaju te jednostavnost kreiranje grafičkih sučelja putem Visual Studio sučelja.

- Jednostavnost i brzina razvoja – Jednostavnost u smislu jednostavnog povezivanja podataka sa komponentom. Nije nam potreban nikakav server koji nam dostavlja podatke kao što je stvar kod Elektrona. Ovdje je moguće i direktno povezivanje sa

bazom što također ubrzava proces razvoja jer ne treba raditi dodatne servise koji bi samo dohvaćali podatke iz baze i slali ih na klijenta.

- Jednostavnost kreiranja prozora – Zahvaljujući grafičkom sučelju koje nudi Visual Studio vrlo jednostavno je odabrati komponentu iz izbornika i samo je dovući na prozor. Postavljanje pozicije komponente i svih dodatnih svojstva komponente je moguće uz pomoć grafičkog sučelja [23].

3.3.2.2. Nedostaci

Velika većina nedostataka Windows forma vezani su uz ograničenost. Zbog pojave web tehnologija razvoj stolnih tehnologija je zastao te iz toga razloga tehnologije za izradu stolnih aplikacija su ograničene. U ovom slučaju nedostaci Windows forma su ograničenost dizajniranja i mijenjanja stilova postojećih komponenti, ne postoji podrška za operacijske sustave koji nisu Windows te nedostatak komponenti koje su podržane.

- Ograničenost dizajna – Kako je već spomenuto Windows forme imaju jednostavnu integraciju komponenti na zaslon, ali kompleksniji dizajn i animiranje nedostatak su Windows formi, za tako nešto preporuča se korištenje WPF-a [23].
- Nema podrške za ostale operacijske sustave – Aplikacije napravljene pomoću Windows formi moguće je pokrenuti samo Windows operacijskom sustavu.

- Nedostatak komponenti – Windows forme imaju određen broj komponenti podržanih odmah iz starta. No ako je potrebna neka posebna komponenta potrebno je uzimanje komponenti od vanjskih proizvođača, a obično se takve komponente trebaju plaćati [23].

3.3.3.WPF

WPF je UI tehnologija za kreiranje stolnih aplikacija. Sam princip rada WPF-a sličan je Windows formama. To jest WPF je podržan od strane Windows formi. Najčešća primjena mu je za aplikacije koje trebaju imati lijepi vizualni izgled i aplikacije koje iz nekog razloga trebaju imati podršku za animacije [23].

3.3.3.1. Prednosti

U odnosu na Windows forme WPF je noviji te iz tog razloga ima više prednosti u odnosu na Windows forme. Neke od prednosti su kontrola nad dizajnom, mogućnost pisanja XAML jezika i poboljšanje performansi u odnosu na Windows forme.

- Kontrola nad dizajnom – Za razliku od Windows formi, WPF pruža mogućnost stvaranja složenog dizajna i složenih animacija [23].

- XAML – Mogućnost korištenja nekog markup jezika uvijek je dobrodošla. Omogućuje korisniku jednostavnije kreiranje vlastitog dizajna [23].

- Performanse – Kako bi se grafički prikaz što efikasnije prikazao na zaslonu WPF dodatno koristi i „hardware“ akceleraciju [23].

- ***Nedostaci***

- Nedostaci vezani uz Windows forme popravljeni su WPF-om. No uvođenjem WPF-a pojavili su se novi nedostaci. Ti nedostaci su nedostatak efikasnosti, ograničena kontrola i nedostatak zajednice jer je tehnologija relativno nova.

- Efikasnost – U odnosu na Windows forme troši više memorije.

- Ograničenost kontrola – U odnosu na Windows forme kontrole su dosta više ograničene.

- Nedostatak zajednice – U odnosu na Elektron i Windows forme postoji manjak zajednice i biblioteka od strane zajednice.

3.4. Komparativna analiza tehnologija

U Poglavlju „Opis tehnologija“ analizirane su i uspoređene tehnologije za izradu korisničkih sučelja, ali napravljena je usporedba i analiza programskih jezika za izradu serverskih aplikacija. Uz analizu i obradu tehnologija i programskih jezika razmotrene su, opisane i uspoređene tehnologije za kreiranje stolnih aplikacija. Za tehnologije za izradu korisničkih sučelja kreirani su i isječci koda koji prikazuju kako izgleda sintaksa pisanja koda. U ovom poglavlju bit će sažeti opis tehnologija i programskih jezika te razlog zašto je odabrana tehnologija dobra za korištenje na ovom projektu.

3.4.1. Tablična usporedba tehnologija za izradu korisničkih sučelja

U ovom poglavlju u tabličnom obliku biti će prikazane glavne karakteristike po kojima bi korisnik bez ikakvog detaljnog znanja o pojedinoj tehnologiji mogao odabrati tehnologiju za izradu korisničkih sučelja. U tablici (Tablica 1.) je prikazano kada je tehnologija izdana. To je pokazatelj koji pokazuje je li tehnologija dovoljno zrela za korištenje u produkcijskoj okolini. Zatim sljedeće je prikazano koji programski jezici se mogu koristiti prilikom kreiranja korisničkih sučelja u pojedinim tehnologijama. Krivulja učenja jedan je od bitnijih pokazatelja prilikom odabira tehnologije. Ako je tehnologija previše kompleksna biti će odbojnija za naučiti u odnosu na one jednostavnije. Ako je krivulja učenja strma to znači da je sam proces učenja tehnologije kompliciraniji i kroz vrijeme postaje sve više kompliciran zbog same kompleksnosti. Pokazatelj primjene tehnologije daje na znanje za kakve projekte je pogodno koristiti tehnologiju. Važno je odmah u početku znati mogućnosti i ograničenja tehnologije kako bi se izbjegli problemi koji bi pristizali u kasnijim fazama razvoja. Pokazatelj veličine pokazuje koliko bazna instalacija tehnologije zauzima memorije, ako su korisniku performanse i memorija važan uvjet onda je potrebno odabrati tehnologiju koja zauzima manje memorije.

Tablica 1. Usporedba tehnologija za izradu korisničkih sučelja [autorska izrada]

	Angular	React	Vue
Prvo izdanje	2010.	2013.	2014.
Programski jezik	TypeScript	TypeScript/JavaScript	TypeScript/JavaScript
Krivulja učenja	Strma	Mala	Mala
Primjena	Velike projekte	Velike/Srednje/Male projekte	Srednje/Male projekte
Veličina	679 kB	2.5kB	34.2kB

Iz opisa tehnologija može se vidjeti koje su tehnologije pogodne za što. React i Vue tehnologije su koje su relativno male po veličini, ali su tehnologije koje ne dolaze sa svim instaliranim alatima. To je jedan od razloga zašto su toliko manje u odnosu na Angular. Bitno je napomenuti kako je Angular solucija za velike projekte dok su React i Vue više usmjereni prema srednjim i manjim projektima. Iz tog razloga Angular je bio eliminiran pri odabiru tehnologije za izradu korisničkih sučelja za ovaj projekt. Pri odabiru između Vue i React tehnologije bila je podrška zajednice. React ima puno veću zajednicu u odnosu na Vue te iz tog razloga React je tehnologija koja je odabrana za ovaj projekt.

3.4.2. Tablična usporedba programskih jezika za izradu serverskih aplikacija

Isto kao i kod tehnologija za izradu korisničkih sučelja u tabličnom obliku (Tablica 2.) prikazane su karakteristike koje opisuju programske jezike bez detaljnog znanja o navedenim programskim jezicima. Prvi podatak u tablici je godina izdanja programskog jezika, može biti korisno za promatranje razvoja jezika kroz godine isto tako prilikom odabira jezika nije poželjno odabrati programski jezik koji je novitet na tržištu. Sljedeći podatak su tehnologije koje su na raspolaganju sa navedenim programskim jezicima, poželjno je da ne postoji niti prevelik niti premali odabir tehnologija. Podatak o krivulji učenja pokazatelj nam je koliko je vremena potrebno kako bi se programski jezik naučio koristiti. Zadnji podatak je sama primjena programskog jezika. Vidljivo je kako su JavaScript i Python primjenjivi na sve oblike projekta dok je C# zbog svoje povećane kompleksnosti primjenjiv na srednje i velike projekte.

Tablica 2. Usporedba programskih jezika za serverske aplikacije [autorski rad]

	JavaScript	Python	C#
Prvo izdanje	1995.	1991.	2000.
Tehnologije	NodeJs, React, Deno, Vue, Angualar	Django, Flask	.NET, .NET Framework, .NET core
Krivulja učenja	Mala	Mala	Srednja
Primjena	Mali/Srednji/Veliki projekti	Mali/Srednji/Veliki projekti	Srednji/Veliki projekti

Svaki od ovih programskih jezika pogodan je za neko područje izrade programskih rješenja. JavaScript i Python popularniji su i mogu se koristiti za sve oblike i veličine projekata. C# na drugu stranu je kompleksniji i pogodan je za projekte koji su malo kompleksniji. C# je odabran za ovaj projekt iz razloga što je već postojeća aplikacija koju treba migrirati napisana u C#.

4. Migracije

U prethodnom poglavlju nabrojane su neke od najpoznatijih i trenutno najmodernijih tehnologija. Analizom i usporedbom nabrojanih tehnologija za razvoj korisničkih sučelja React se pokazao kao najbolja opcija. Također je utvrđeno kako za serverski dio nema smisla mijenjati programski jezik jer je cijela poslovna logika i obrada podataka napravljena u C# programskom jeziku koji se pokazao najboljim od navedenih programskih jezika.

Migracija aplikacija je proces premještanja aplikacija iz jednog računalnog okruženja u drugo računalno okruženje [24]. Postoji više oblika migracija. Neki od oblika su migracija su migracija aplikacije iz vlastitog privatnog lokalnog okruženja na neku cloud platformu (npr. migriranje aplikacije s vlastitog servera na Azure ili AWS platformu), zatim migracija aplikacije iz jedne tehnologije na drugu tehnologiju (npr. migriranje aplikacije pisane u Angularu u React tehnologiju), slično prethodnome migracija aplikacije iz jednog programskog jezika u drugi (npr. migracija aplikacije pisane u JavaScript programskom jeziku u C# programski jezik) te migracija aplikacije sa jedne platforme na drugu platformu (npr. migracija stolne aplikacije u web aplikaciju). Za cilj ovog završnog rada koristiti će se migracija aplikacije sa jedne platforme na drugu platformu.

4.1. Zašto migrirati sa stolne aplikacije na web?

Do potrebe migracije može doći iz nekoliko razloga. Web aplikacije dostupne su sa bilo kojeg uređaja koji ima pristup internetu, dok za stolne aplikacije potrebni je imati računalo koje ima instaliranu aplikaciju. Za web aplikaciju nije potrebna instalacija, moguće je momentalno korištenje. Sljedeći razlog je ograničenost od strane komponenti unutar računala, kako su stolne aplikacije zahtjevnije za pokrenuti potrebno je ulaganje u bolje komponente. Stolne aplikacije moraju biti instalirane na računalo kako bi korisnici mogli pristupiti podacima dok web aplikacije su dostupne sa bilo kojeg računala bez potrebnih dodatnih instalacija [25]. Jedan od kritičnijih razloga migriranja je veći napredak web tehnologija u odnosu na tehnologije za izradu stolnih aplikacija. Slika 4 prikazuje koliko su web aplikacije naprednije i pogodnije od stolnih aplikacija.

Category	Desktop Application	Web Application
Installation	Needs installing and requires the OS to meet certain conditions.	No installation is needed; launched by accessing a URL from a browser.
Security	Especially if it's a monitored enterprise device, the desktop usually has a lot of security and protection measures, and therefore it's considered to be more secure. But residual files might remain on such a system even after uninstallation. Chromium-based applications (SSB/PWA) might be vulnerable to browser exploits.	As the browser is the most vulnerable app, a web application is considered less secure.
Automatic Updates	Must manually download and install updates.	Web apps update themselves automatically.
Development	The app must be ported for each operating system to reach all users.	Developed for universal delivery method—a browser—that works on any OS.
Usability	Can be launched independently of other applications, but information can't be easily shared with remote users in most cases.	It's quick and easy to start working wherever, whenever. Easy, collaborative file sharing with other users and ability to readily update users' changes.
Cross-Platform Availability	Can only be accessed on the specific device where the app was installed.	Can be launched from any device equipped with a web browser.
Mobile Access	In many cases, the app lacks a mobile version.	Most web applications can also operate on mobile devices.
Hardware Resources Use	Uses more endpoint processing power than a web app.	Web services require significantly less processing power than desktop apps, and don't overload disk storage since data is saved in the cloud.
Internet Dependency	In most cases, doesn't require an internet connection. The internet is accessed only for version updates or syncing with the app server.	Needs constant internet connectivity. Can function in offline mode, but many features are disabled. Moreover, a weak or intermittent internet connection can negatively affect performance.
Deployment	Requires installation on every device.	Only a browser is required.
User Experience	It's much easier to locate a desktop application in use.	Users need to remember in which browser tab the app is running. Closing all tabs also closes the app.

Slika 4. Usporedba stolnih aplikacija i web aplikacija [25]

Iz Slike 4 moguće je vidjeti kako web aplikacije imaju veće prednosti u odnosu na stolne aplikacije. Neke od značajnijih prednosti su lakši inicijalni korak korištenja aplikacije. Pod tim se smatra da nema potrebe kontaktirati osobu u prodaji koja naknadno pošalje stolnu aplikaciju

koja se mora instalirati. Web aplikaciju je moguće koristiti odmah nakon prijave ili registracije. Zatim dijeljenje aplikacije sa drugima je lakše jer za web aplikaciju potrebno je samo podijeliti URL aplikacije dok stolnu aplikaciju nije moguće podijeliti bez da je druga osoba instalira na svoje računalo. Prednost web aplikacije je također to što web aplikacije nije potrebno instalirati na svoje računalo što znatno ubrzava početak korištenja aplikacije.

4.2. Proces migriranja

Migracija aplikacija prema web mjestu vmware [24] provodi se kroz četiri koraka. Prvi korak je planiranje migracije, zatim testiranje migracija na testnom projektu, migriranje u valovima i dokumentiranje procesa te je na kraju potrebno odraditi završno testiranje.

4.2.1. Planiranje migracije

Prilikom planiranja potrebno je analizirati trenutnu aplikaciju i odrediti kompleksnost trenutnog rješenja. Uz određivanje kompleksnosti ovo je korak u kojem se treba analizirati tim koji će vršiti migraciju, vrlo bitno kako bi se što preciznije planirala migracija. Također dobro je razmotriti već postojeće alate koji će olakšati proces migracije kako bi se uštedilo što više vremena i resursa. Neki od tih alata mogu biti za prijenos podataka s jedne platforme na drugu [24].

4.2.2. Testiranje migracija na probnim projektima

Prije provedbe prave migracije predloženo je da se migracija provede na nekom probnom projektu. Na probnom projektu izvršiti valove migracije i nakon svakog vala migracija potrebno je aplikaciju testirati. Ako postoje problemi potrebno ih je riješiti prije provedbe novog vala migracija. Na taj način minimizira se mogućnost pogreške [24].

4.2.3. Provedba migracije i dokumentiranje

Nakon provedbe uspješne testne migracije kreće se na migriranje stvarne aplikacije. Migraciju je preporučeno izvesti u valovima, to jest dio po dio. Nakon provedbe svakog vala migracije potrebno je dokumentirati rezultate. Na ovaj način izbjegavaju se greške i mogući sukobi u kodu [24].

4.2.4. Završno testiranje

Nakon provedbe migracije cjelokupne aplikacije potrebno je izvršiti završne testove koji testiraju stabilnost, performanse i sigurnost web aplikacije. Ako je sve prošlo bez problema aplikacija se može pustiti u produkciju i dati korisnicima na korištenje, ali ako se pojave problemi potrebno je te probleme popraviti i ponoviti migraciju u kojoj je došlo do problema.

4.3. Elementi migracije

Kako bi stolna aplikacija uspješno bila migrirana na web potrebno je proći kroz proces migriranja takvih aplikacija. Generalno se aplikacija sastoji od 3 sloja. To su sloj rada sa podacima, sloj poslovne logike i sloj korisničkog sučelja. Svaki od ta tri sloja biti će opisan te će biti objašnjeno zašto ga je potrebno ili nije potrebno migrirati.

4.3.1. Migracija korisničkog sučelja

Najčešći razlog migriranja stolnih aplikacije na web je korisničko sučelje [26]. Ako se gleda sa nekog stajališta tehnologije to je također i najpromjenjiviji sloj aplikacije. Do promjena dolazi najčešće zbog zahtjeva korisnika, ali isto tako i zbog brzog napretka tehnologije zaslužene za kreiranje korisničkih sučelja [26]. Mogućnost ponovne upotrebe koda za kreiranje korisničkih sučelja vrlo je mala. Razlog toga je promjena platforme, ali isto tako i programskog jezika.

Bitno je napomenuti da kod nije jedina stvar koja se migrira prilikom migriranja korisničkih sučelja. Prilikom migracije korisnički sučelja uz migraciju vizualnog dizajna migriramo i interakcije, tijek korisničkog sučelja, elemente koji se koriste u sučelju, na koji način pristupamo tim elementima, validaciju korisničkog unosa, logiku koja se provodi nad komponentama [26]. Bitno je prilikom migracije obratiti pažnju na zadržavanje protoka korištenja sučelja zbog navika korisnika ili ako korisnici imaju povratnu informaciju vezanu uz probleme korištenja sučelje ovo je prava prilika za popraviti te probleme.

Ovisno o korisničkoj bazi i kompleksnosti korisničkog sučelja postoji mogućnosti da se korisničko sučelje prilagodi za mobilne uređaje kako bi aplikacija bila još dostupnija [26].

4.3.2. Migracija poslovne logike

Poslovna logika su prilagođena pravila ili algoritmi koji upravljaju razmjenom informacija između baze podataka i korisničkog sučelja [27]. Korisničko sučelje stolnih aplikacija, poslovna logika i integracija sa bazom podataka nalaze se u jednom projektu te se sa korisničkog sučelja može direktno pristupati bazi podataka i metodama poslovne logike, ali situacija sa web aplikacijom je drugačija. Korisničko sučelje web aplikacije odvojeno je od

poslovne logike i baze te je potrebno odrediti drugačiji pristup metodama poslovne logike. Iz ovog razloga potrebna je migracija poslovne logike.

Kako bi se preko korisničkog sučelja moglo pristupati metodama poslovne logike potrebno je te metode izložiti na internet preko HTTP-a. U posljednje vrijeme pristup uslugama temeljen na REST-u postajo je jako popularan [26]. Kako bi se te metode mogle izložiti preko HTTP-a potrebno je kreirati novi servis koji omogućava komunikaciju između korisničkog sučelja i poslovne logike. Taj servis naziva se API. API zatim na zahtjev sa strane korisničkog sučelja na serveru dohvati podatke koje zatim prolaze kroz poslovnu logiku i nakon toga obrađene podatke vraća na stranu korisničkog sučelja kako bi se ti podaci prikazali klijentu. Zahtjevi kojima se komunicira između sučelja i servera nazivaju se HTTP zahtjevi te mogu biti GET, POST, PUT, DELETE i PATCH tipa. Svaka od tih metoda označava API servisu o kakvom se zahtjevu radi te API na temelju tih metoda vraća podatke, uređuje zapis u bazi, kreira zapis u bazu ili briše zapis iz baze.

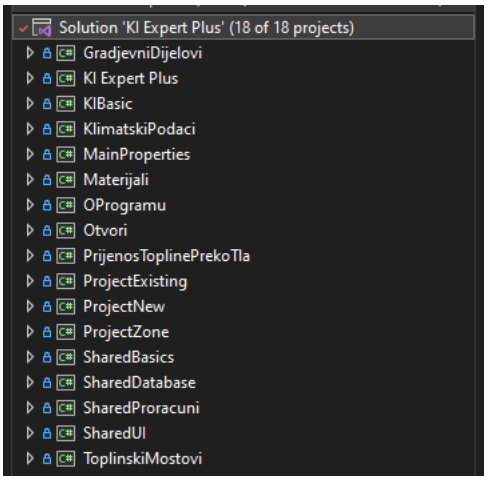
4.3.3. Migracija rada s podacima

Za razliku od sloja korisničkog sučelja i sloja poslovne logike sloj podataka je najmanje promjenjiv [26]. Upravo iz tog razloga taj sloj može biti korišten više puta bez dodatnih promjena. Do potrebe za migracijom dolazi ako je potrebno podržati neke moderne web standarde ili ako se stolna aplikacija oslanjala na pohranjene procedure ili izravno izvršavanje naredbi i upita na bazu podataka [26]. U tom slučaju preporuča se te naredbe i upite preseliti u posebne servise kako bi se tim metodama i upitima moglo pristupiti preko korisničkog sučelja putem HTTP zahtjeva.

5. Migracija postojećeg programskog proizvoda

U ovom poglavlju opisan je postupak migracije .NET Windows Forms stolne aplikacije u web aplikaciju. U poglavlju će se ukratko napraviti analiza trenutnog rješenja, zatim kako je migracija planirana i na kraju će biti opisan postupak same provedbe migracije. Buduća aplikacija biti će preslika trenutne aplikacije u smislu dizajna i funkcionalnosti, ali implementacija tih funkcionalnosti na korisničkom sučelju i serverskom dijelu razlikovati će se od trenutnog rješenja.

Rješenje je pisano na .NET framework 4.7.2 verziji. Riječ je o kompleksnoj aplikaciji za kreiranje i upravljanje projektima. Aplikacija nema prijave što znači da ne postoji korisnik. Baza koja se koristi u projektu je SQLite baza podataka. Važno je napomenuti kako ne postoji jedna klasična baza sa podacima nego svaki projekt je baza za sebe (posebna SQLite datoteka).



Slika 5. Struktura rješenja prije migracije

Rješenje se sastoji od 17 projekata koji se nalaze unutar jednog rješenja. Iz razloga što je aplikacija veoma kompleksna i opširna u ovom radu se ne koriste svi projekti, nego samo oni neophodni za dohvaćanje postojećih projekata i kreiranja novog projekta. Projekt koji će se najviše koristit je SharedDatabase iz razloga jer su sve operacije nad bazom podataka u tom projektu. Migracija podatkovnog sloja neće biti potrebna jer metode za bazu podataka koje su korištene za stolnu aplikaciju vrijede i za web aplikaciju. Korisničko sučelje biti će kreirano ispočetka jer Windows Forme ne mogu biti pokrenute u web pregledniku. Poslovna logika morat će biti dodatno prepravljena jer se u nekim metodama koriste metode za rad nad formama, što nam nije potrebno za web aplikaciju.

5.1. Pregled postojećeg rješenja

Kako je već navedeno u prethodnom odlomku stolna aplikacija napisana je u .NET framework 4.7.2 verziji. Slika 5. prikazuje strukturu projekata unutar rješenja. Trenutna verzija aplikacije u nekim segmentima miješa poslovnu logiku sa logikom koja se odnosi na forme što je problem jer se te metode kasnije ne mogu pozivati. Razlog te nemogućnosti pozivanja već postojećih metoda je pogreška koju program baca jer ne postoji kreirana forma. Isječak koda 4. prikazuje kako je na nekim mjestima miješana poslovna logika i logika koja se koristi za forme.

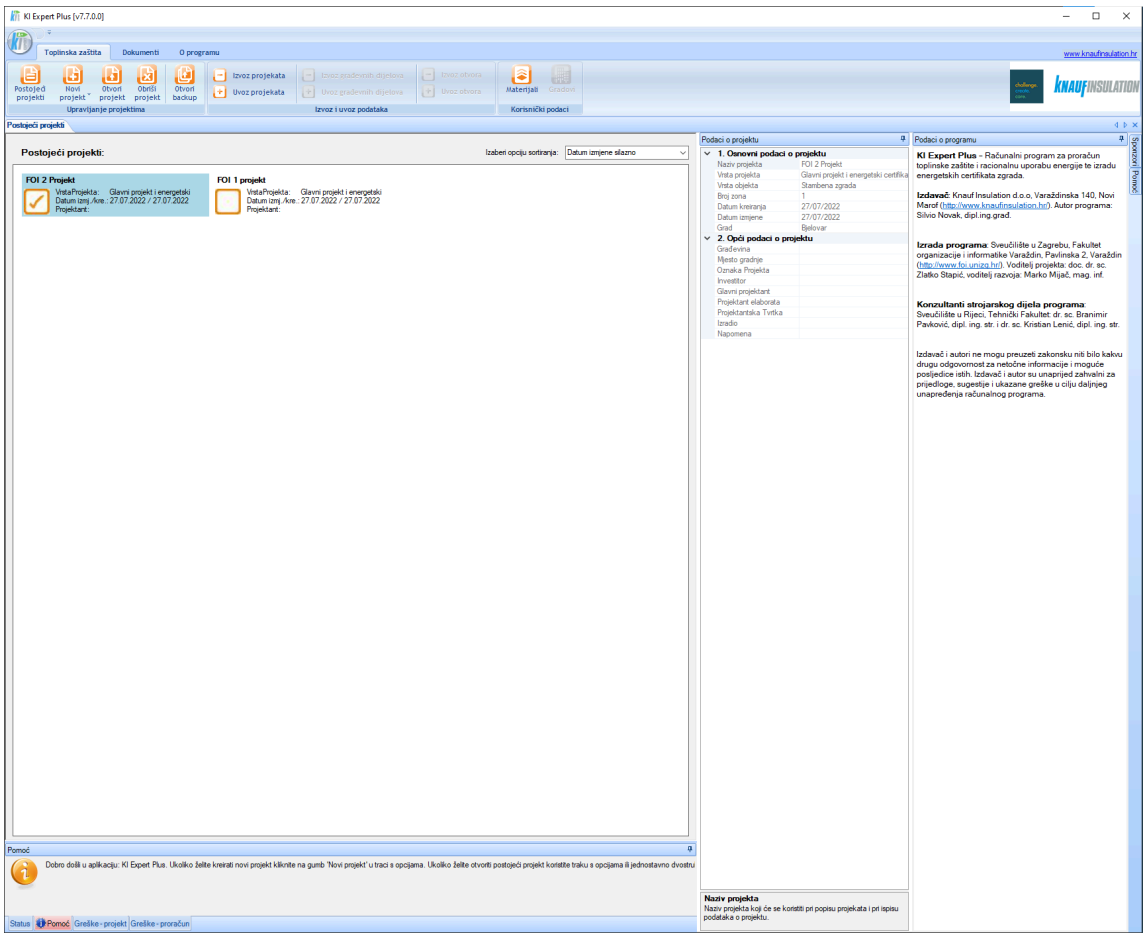
```
private void CreateNewProject(VrsetProjekta vrstaProjekta)
        {
            if ((SharedGlobals.Instance.ApplicationState != 
SharedGlobals.AppState.ApplicationImportingProjects) || 
(GetAllWorkingProjects().Length == 0))
            {
                //priprema baze novog projekta
                string projectPath = ManageDocs.CreateProjectFromTemplate();

                //otvaranje konekcije na bazu i compact baze
                ManageDB.DataBasePath = projectPath;
                ManageDB.CompactDatabase();

                //Kreiranje novog projekta u bazi
                ManageDB.CreateNewProject();
                ManageDB.ExecuteNonQuery("UPDATE Projekt SET IdVrsteProjekta = " + 
(int)vrstaProjekta);
                ManageDB.ExecuteNonQuery("UPDATE Environment SET 
VerzijaZadnjegSpremanja = VerzijaBaze");

                KIBasic.Events.OnProjectInitializing();

                //promjena ribbona

RibbonControl.Instance.SetVisibleRibbon(RibbonControl.Ribbons.RibbonProject);

                SharedGlobals.Instance.ApplicationInsideProject = true;
                //pokretanje novog projekta
                frmMain.Instance.SetUserInterface(uiNewProject.Instance);

                //osvježavanje postavki dokumenta ispisa
                //Settings.Instance.OutputFile = "";
            }
            else
            {
                DialogKIOk dialogOK = new DialogKIOk();
```

```
                dialogOK.ShowDialog("Upozorenje!", "Nije moguće kreirati novi
projekt budući da je uvoz projekata u tijeku!" + Environment.NewLine + "Završite
uvoz projekata i pokušajte ponovno!", DialogIcons.Warning);
            }
        }
```

Isječak koda 4. Prikaz miješanja poslovne logike i logike za forme [autorska izrada]

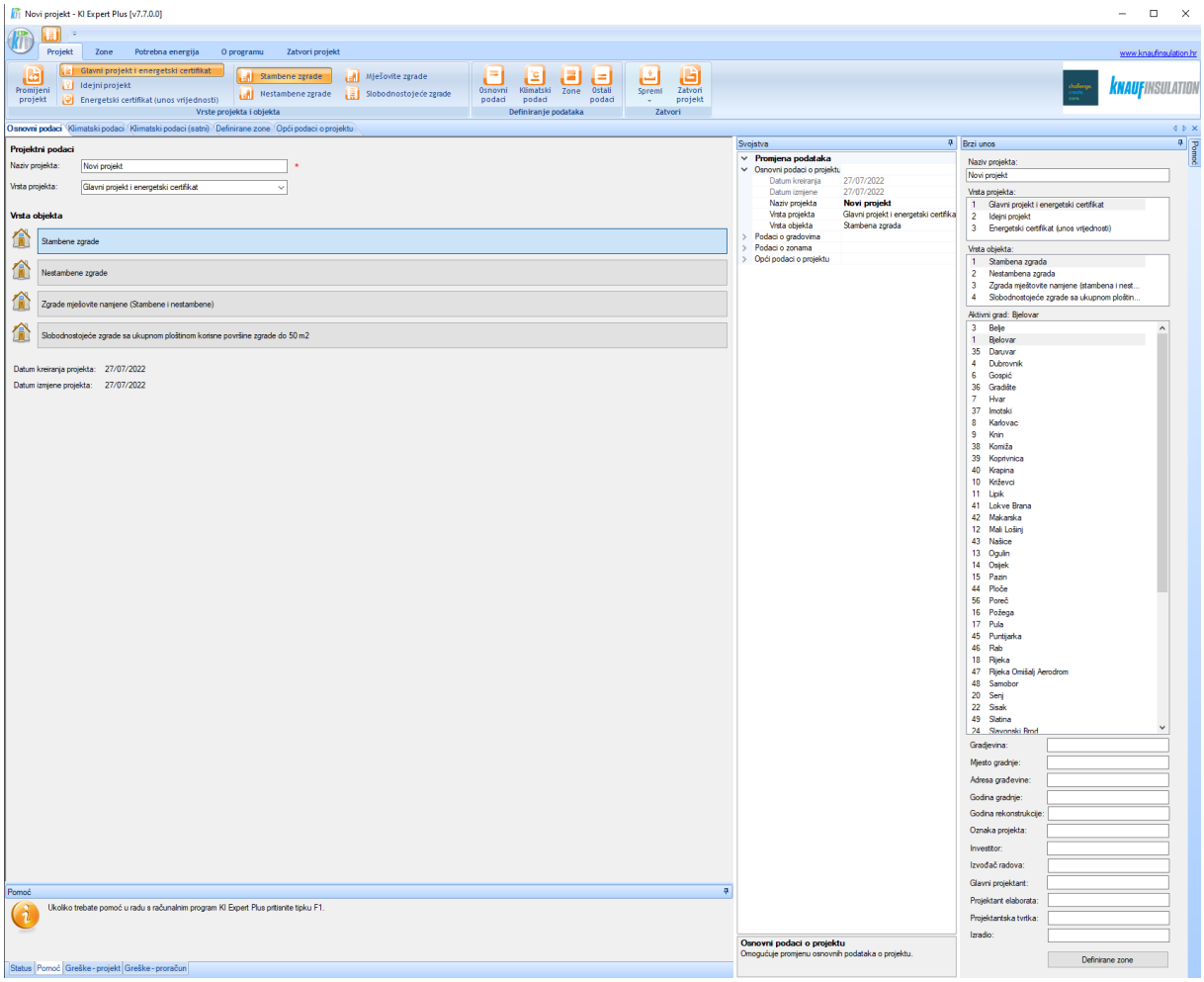
RibbonControl.Instance.SetVisible je metoda koja služi za rad sa formama i zbog te metode nije moguće koristiti metodu „CreateNewProject“ te je potrebno kreirati novu klasu u koju bi se ta poslovna logika prekopirala bez logike koja se koristi za forme. Ovo nije jedina metoda koja miješa poslovnu logiku i logiku za forme, ovo je samo jedan od primjera. Za sve takve metode bit će potrebno dijelove tih metoda izdvojit u nove metode u novim klasama.

Sljedeća bitna stvar je korisničko sučelje i njegov trenutni izgled. Korisničko sučelje je također dosta kompleksno te se sastoji od velike količine elemenata. Primjer projekta prilikom pokretanja same aplikacije može se vidjeti na Slici 6. na kojoj su prikazani svi postojeći projekti.



Slika 6. Prikaz početnog zaslona aplikacije

Na ovom zaslonu moguć je pregled trenutno postojećih projekata, brisanje tih projekata, otvaranje i mijenjanje projekata, kreiranje novih projekata te uvoz i izvoz projekata. Na sljedećoj slici, Slika 7. prikazan je prikaz kada se pritisne na gumb „Novi projekt“.



Slika 7. Prikaz kreiranja novog projekta

Slika 7. prikazuje kako izgleda početno korisničko sučelje koje se otvara prilikom kreiranja novog projekta. Kao što je moguće vidjeti na slici postoji velik broj kartica i dodatnih prozora, za svrhu ovog rada velika većina tih prozora nije bitna. Za ovaj rad bitno je da se kreiranje projekta realizira pomoću kartice „Brzi unos“ iz razloga što su to neki od jednostavnijih detalja o projektu.

5.2. Plan migracija

U planu migracija biti će opisan zamišljeni protok migriranja aplikacije. Proći će se kroz sva tri sloja aplikacije i opisati koji su sve procesi zamišljeni za pojedine slojeve. Biti će također prikazane metode koje se koriste u pojedinim slojevima kako bi se izvršile naredbe koje se šalju sa novog korisničkog sučelja.

5.2.1. Migracija podatkovnog sloja

Podatkovni sloj služi kako bi se lakše pristupalo operacijama nad bazom podataka. Web aplikacija i stolna aplikacija imaju istu funkcionalnost te iz tog razloga nema potrebe mijenjati strukturu baze ili način upisivanja i čitanja u bazu podataka. Zbog toga u podatkovnom sloju imamo migraciju bez promjena metoda i klasa, sve metode koje se izvršavaju nad bazom podataka odgovaraju za integraciju sa novom web aplikacijom. Te metode najviše se koriste u sloju poslovne logike za spremanje zapisa u bazu ili čitanje podataka iz baze. Plan je zapravo koristiti sve već postojeće metode koje postoje na podatkovnom sloju bez ikakvih promjena.

5.2.2. Migracija poslovne logike

Za razliku od podatkovnog sloja gdje se provodi migracija bez promjena metoda i klasa, kod sloja poslovne logike potrebno je migrirati metode i klase. Analizom trenutnog koda bilo je vidljivo kako je dio poslovne logike pomiješan sa logikom koja je povezana na korisničko sučelje. Upravo to miješanje poslovne logike i logike korisničkog sučelja najveći je problem prilikom korištenja već postojećih metoda. Problem je to što metoda ne dobiva potrebu referencu na formu i zbog toga se događaju greške. Iz tog razloga plan za poslovnu logiku je izdvajanje poslovne logike i samo poslovne logike u nove metode koje bi se zatim koristile umjesto postojećih. Također do sad u stolnoj aplikaciji korisničko sučelje imalo je direktan pristup metodama poslovne logike, ali u slučaju web aplikacije to nije moguće. Plan da se to riješi je kreiranje upravljača iliti kontrolera u kojima bi se izvršavale novokreirane metode sa izdvojenom poslovnom logikom, bez logike korisničkog sučelja. Zatim bi se sa web korisničkog sučelja putem HTTP zahtjeva koji bi se obradili u upravljaču mogli dohvatiti svi potrebni podaci, ali isto tako izvršiti naredbe kreiranja i brisanja.

5.2.3. Migracija korisničkog sučelja

Kako bi se stolna aplikacija uspješno preselila na web aplikaciju potrebno je cijelu aplikaciju napisati iznova u jednoj od web tehnologija za kreiranje korisničkih sučelja, u našem slučaju React aplikacija. Istraživanjem je zaključeno kako se Windows Forms korisničko sučelje ne može nikako iskoristiti za web, ali zato je moguće iskoristiti dizajn i protok kretanja

korisnika kroz aplikaciju. Plan migracije korisničkog sučelja je kreirati novu aplikaciju u React tehnologiji uz zadržavanje trenutnog dizajna i načina rada aplikacije korisničkog sučelja. Način na koji će se to izvesti nije isti kao kod Windows Formi, ali će zato finalni rezultat biti jako sličan trenutnom rješenju.

5.3. Provedba migracija

Nakon određivanja plana u prethodnom poglavlju taj plan potrebno je realizirati. U ovom poglavlju biti će opisano kako su migracije provedene, koje metode je trebalo maknuti, a koje trebalo promijeniti i nadodati. Noviteti će biti potkrijepljeni isječcima koda kako bi se vizualno vidjelo što je i kako je nadodano. Kao i kod planiranja migracije biti će opisani svi slojevi.

5.3.1. Migracija podatkovnog sloja

Kako je i bilo planirano podatkovni sloj nije bilo potrebno mijenjati pri migraciji. Sve korištene metode osim jedne metode rade bez ikakvih dodatnih promjena. U jednu metodu bilo je potrebno dodati dodatan kod. Riječ je o metodi koja briše datoteku, jer je riječ o SQLite bazi podataka i tome da je svaki projekt baza sa sebe. Kako bismo obrisali projekt potrebno je obrisati datoteku koja predstavlja taj projekt. Isječak koda 5. prikazuje stari izgled metode za brisanje projekta, dok isječak koda 6. prikazuje trenutni izgled te metode.

```
public bool DeleteFile(FileInfo file, bool saveRecovery = false)
        {
            try
            {
                if (saveRecovery)
                {
                    ZipItRecoveryFile(file);
                }

                file.Delete();
                file = null;

                return true;
            }
            catch
            {
                return false;
            }
        }
```

Isječak koda 5. Stari izgled metode za brisanje projekta [autorski rad]

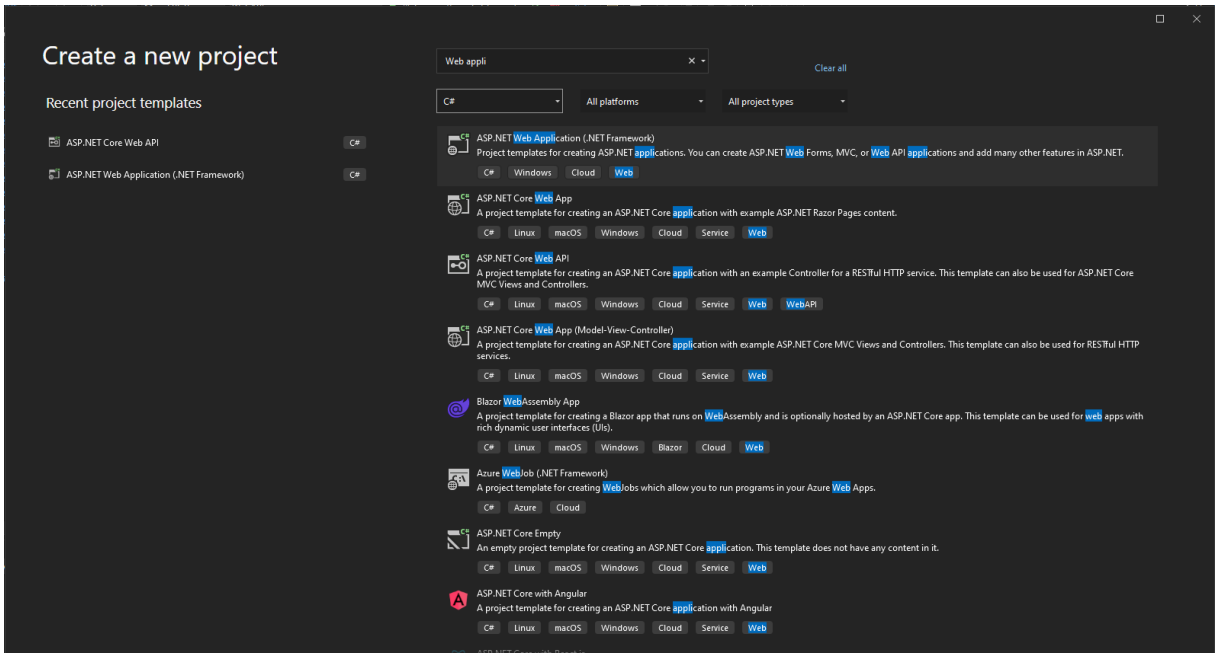
```
public bool DeleteFile(FileInfo file, bool saveRecovery = false)
        {
            try
            {
                if (saveRecovery)
                {
                    ZipItRecoveryFile(file);
                }


                System.GC.Collect();
                System.GC.WaitForPendingFinalizers();

                file.Delete();
                file = null;
                return true;
            }
            catch
            {
                return false;
            }
        }
```

Isječak koda 6. Trenutni izgled metode za brisanje projekta [autorski rad]

Razlika između starog oblika i novog oblika *DeleteFile* metode su dvije linije koda. To su metode *System.GC.Collect();* i *System.GC.WaitForPendingFinalizers*. Te metode vezane su uz problem na koji sam naišao prilikom pozivanja *DeleteFile* metode iz upravljača. Problem je bio da je stalno ostajao jedan klijent povezan na datoteku te ostali klijenti nisu mogli pristupati drugim datotekama. Te dvije linije koda riješile su taj problem.

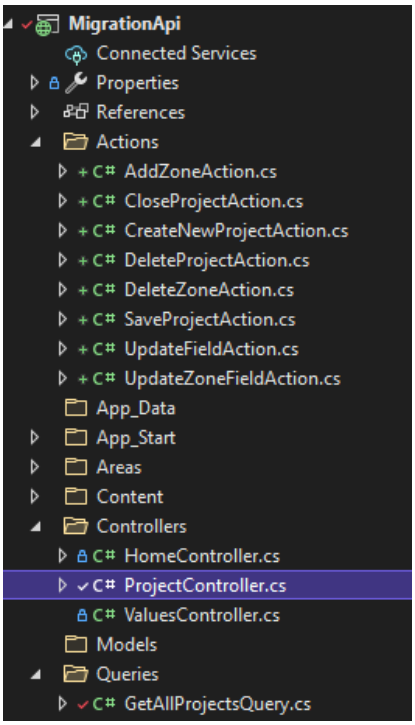
5.3.2. Migracija poslovne logike

Prvi korak pri migraciji poslovne logike sa stolne aplikacije na web aplikaciju bio je kreiranje novog projekta. U Visual Studio programu potrebno je slijediti sljedeće postupke. **File > New > Project** i zatim se otvara prozor prikazan na slici 8.



Slika 8. Odabir novog projekta [autorski rad]

Kako bi se kreirao odgovarajući projekt potrebno je odabrati *ASP .NET Web Application (.NET Framework)* verziju projekta, ako se odabere *ASP .NET Core Web App* dolazi do problema sa Windows Formama jer Windows Forme nisu podržane u toj verziji. Nakon kreiranja projekta automatski se kreira struktura projekta koja je vidljiva na slici 9.



Slika 9. Struktura novog projekta [autorski rad]

Prilikom kreiranja projekta dobiva se projekt strukturiran na način kako je prikazano na slici 9. Kako je riječ o MVC projektu prilikom kreiranja projekta dobivaju se i datoteke za kreiranje korisničkih sučelja. No nama je jedino bitna datoteka *Controllers*. U toj datoteci naknadno je kreiran kontroler pod nazivom *ProjectController.cs.* Taj kontroler sadrži metode preko kojih sa web aplikacije to jest korisničkog sučelja pristupamo poslovnoj logici i podacima projekata.
Prva metoda koja se nalazi u upravljaču služi nam za dohvaćanje trenutnih projekata. Prvi korak bio je kreirati novu statičnu klasu pod nazivom *GetAllProjectsQuery.cs* u koju se odvojila logika za dohvaćanje svih trenutnih projekata. Izgled metode za dohvaćanje tih projekata prikazan je kao Isječak koda 7.

```
public static class GetAllProjectsQuery
    {
        public static List<dynamic> GetAllProjects()
        {
            ManageDocs MD = ManageDocs.Instance();
            string projectDir =
MD.ReturnDirectoryPath(ManageDocsDirectoryEnum.Projects);

            DirectoryInfo di = new DirectoryInfo(projectDir);
            FileInfo[] rgFiles = di.GetFiles("*" +
BasicDictionaries.FileExtensionValues[FileExtensions.KIExpertProject]);

            var projects = new List<dynamic>();
            foreach (FileInfo fInfo in rgFiles)
            {
                if (fInfo.Length <= 100) continue;
                ManageDB.DataBasePath = fInfo.FullName;
                DbDataReader projectData = ManageDB.GetProjectData();

                if (projectData.HasRows)
                {
                    comProjekt com = new comProjekt(projectData);

                    com.ProjectDatabase = fInfo;

                    var temp = new
                    {
                        Putanja = fInfo.FullName,
                        NazivProjekta = com.NazivProjekta,
                        Datum = com.Datum,
                        Projektant = com.Projektant,
                        VrstaProjekta = com.VrstaProjekta

                    };

                    projects.Add(temp);
```

```
                projectData.Close();
                ManageDB.CloseConn();
            }
            else
            {
                //zatvaranje datareadera
                if (!projectData.IsClosed)
                    projectData.Close();
                string fileName = ManageDB.DataBasePath;
                ManageDB.CloseConn();
                ManageDocs.Instance().DeleteFile(fileName);
            }
        }



        return projects;

    }
}
```

Isječak koda 7. Prikaz metode GetAllExistingProjectsQuery [autorski rad]

Cijela klasa sadrži jednu metodu te su klasa i metoda statičke jer ne trebaju inicijalizaciju. Ukratko što se radi u ovoj metodi. Prvo dobivamo putanju direktorija u kojem su smješteni projekti, zatim iz tog direktorija uzimamo sve datoteke koje imaju .kipfx za oznaku tipa datoteke. Zatim prolazimo kroz te datoteke te u „*ManageDB.DatabasePath = fInfo.FullName*“ otvaramo konekciju na bazu i zatim dobivamo podatke o projektu za kojeg je odabrana trenutna datoteka. Te podatke dobivamo uz pomoć *comProjekt* klase. Zatim kreiramo privremeni anoniman objekt u kojeg stavimo samo željene vrijednosti koje će nam se vraćati i zatim taj anoniman objekt stavljamo u listu. Vraćena lista zatim se prosljeđuje na web aplikaciju unutar kontrolera kako je prikazano u isječku koda 8.

```
[HttpGet()]

 [System.Web.Http.Route("api/get-projects")]

        public IHttpActionResult GetProjects()
        {
            try
            {
                var projects = GetAllProjectsQuery.GetAllProjects();
                return Ok(projects);
            }
            catch (Exception ex)
            {
```

```
                return BadRequest(ex.Message);
            }

        }
```

Isječak koda 8. Prikaz kontrolera za dohvaćanje projekta [autorski rad]

Isječak koda 8 prikazuje izgled jedne metode upravljača. U ovom slučaju to je metoda za dohvaćanje svih projekata. Struktura metode vrlo je jednostavna. Iznad samog naziva metode imamo dva atributa od kojih jedan označava putanju preko koje je moguće pozvati ovu metodu dok drugi označava o kakvom se tipu metode radi. Naziv same metode nije bitan, u ovom slučaju naziv je *GetProjects*. Kada klijent napravi zahtjev na ovaj upravljač na web aplikaciju se podaci šalju u formatu kako je prikazano u isječku koda 9.


```
[
  {
    "Name": "FOI 1",
    "Id": 1,
    "ProjectFile": {
      "OriginalPath": "project_202207251849226558.kipfx",
      "FullPath":
"C:\\Users\\PC\\Desktop\\Zavrsni\\SharedDatabase\\Database\\Projects\\project_20220
7251849226558.kipfx"
    },
    "Date": "2022-07-30T14:33:16.0648189+02:00",
    "ProjectType": "Glavni projekt i energetski certifikat"
  },
  {
    "Name": "FOI 2",
    "Id": 1,
    "ProjectFile": {
      "OriginalPath": "project_202207281733288108.kipfx",
      "FullPath":
"C:\\Users\\PC\\Desktop\\Zavrsni\\SharedDatabase\\Database\\Projects\\project_20220
7281733288108.kipfx"
    },
    "Date": "2022-07-28T17:33:28.8759797+02:00",
    "ProjectType": "Glavni projekt i energetski certifikat"
  }
]
```


Isječak koda 9. Struktura podataka koji se šalju na web aplikaciju [autorski rad]

Ovako izgledaju podaci koji se šalju na web aplikaciju kada korisnik želi dohvatiti sve projekte. Ti podaci su nam potrebni kako bi na korisničkom sučelju mogli prikazat ime i datum kreiranja projekta, ali isto tako kako bismo mogli obrisati ili urediti označeni projekt.

Sljedeća metoda koja je je migrirana služi za brisanje projekata. Isto kao i za dobivanje svih projekata metoda za brisanje projekata nalazi se unutar upravljača *ProjectController.* Kako bismo odvojili logiku za brisanje u novu metodu kreirana je nova statična klasa pod nazivom *DeleteProjectAction.cs* Unutar te klase također imamo jednu statičnu metodu *DeleteProject. DeleteProject* metoda prikazana je u isječku koda 10.

```
public class DeleteProjectAction
    {
        public static void DeleteProject(string path)
        {

            string fullPath =
$"../SharedDatabase\\Database\\Projects\\{path}.kipfx";

            ManageDocs md = ManageDocs.Instance();

            md.DeleteFile(fullPath);

            string backupPath =
md.ReturnDirectoryPath(ManageDocsDirectoryEnum.BackUp) + "\\" + $"{path}.kipfx";

            FileInfo[] backups = md.ShowBackupFiles(backupPath);
            foreach (FileInfo file in backups)
            {
                md.DeleteFile(file.FullName);
            }

            md.DeleteDirectory(backupPath);

        }
    }
```

Isječak koda 10. Prikaz metode DeleteProject [autorski rad]

Iz koda vidljivo je kako se u metodu prosljeđuje parametar iz upravljača. Taj parametar je ime datoteke projekta. *md.DeleteFile(fullPath)* je metoda koja je zaslužna za brisanje projekta koji je odabran na strani korisničkog sučelja. Također postoji i opcija brisanja backup projekta na način da se prvo dobi putanja do tog projekta i zatim se taj projekt obriše na isti način kao i originalni projekt. Metoda se poziva u kontroleru na način koji je pokazan u isječku koda 11.

```
[HttpDelete()]
        [System.Web.Http.Route("api/delete-project/{path}")]
        public IHttpActionResult DeleteProject(string path)
        {
            try
            {

                DeleteProjectAction.DeleteProject(path);
                return Ok("Successfully deleted the project!");
            }
            catch(Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
```

Isječak koda 11. Prikaz kontrolera za brisanje projekta [autorski rad]

U kontroleru za brisanje projekta naziv same datoteke, to jest naziv datoteke projekta šalje se kao dio putanje. To je moguće vidjeti u „Route“ dijelu koda gdje se na standardnu putanju do upravljača u vitičastim zagradama još dodaje taj „path“ dio. To je samo ime projekta. Zatim se taj naziv projekta prosljeđuje u servis koji je zaslužan za brisanje projekta te se unutar servisa obriše taj cijeli projekt.

Sljedeća migrirana metoda je metoda za pripremu i kreiranje projekta. Isto kao i prethodne metode potrebno je bilo kreirati novu metodu statičku metodu pod nazivom *CreateNewProject* u novoj statičkoj klasi *CreateNewProjectAction*. Prikaz nove metode prikazan je na isječku koda 12.

```
    public class CreateNewProjectAction
    {
        public static void CreateNewProject()
        {
            var md = ManageDocs.Instance();
            FileInfo[] workingProjects =
md.ShowProjects(md.ReturnDirectoryPath(ManageDocsDirectoryEnum.Working));

            if ((SharedGlobals.Instance.ApplicationState !=
SharedGlobals.AppState.ApplicationImportingProjects) || (workingProjects.Length ==
0))
```

```
            {
                string projectPath = md.CreateProjectFromTemplate();

                ManageDB.DataBasePath = projectPath;
                ManageDB.CompactDatabase();


                ManageDB.CreateNewProject();
                ManageDB.ExecuteNonQuery("UPDATE Projekt SET IdVrsteProjekta = " +
1);
                ManageDB.ExecuteNonQuery("UPDATE Environment SET
VerzijaZadnjegSpremanja = VerzijaBaze");

                KIBasic.Events.OnProjectInitializing();

                Zona currentZona = SharedGlobals.Instance.ActiveZona as Zona;
                if (currentZona == null)
                {
                    currentZona = ZonaCollection.Instance[0] as Zona;
                }

                SharedGlobals.Instance.ApplicationInsideProject = true;

                md.saveProject(projectPath);
            }
        }
    }
```

Isječak koda 12. Prikaz metode CreateNewProject [autorski rad]

Prvo što se u ovoj metodi provjerava je ako postoje projekti koji su u „Working“ statusu. Što znači da su otvoreni, ali nisu spremljeni. Zatim ako ne postoje uz pomoć metode „*md.CreateProjectFromTemplate()“* iz predloška se kreira novi projekt. Na način da se iz predloška baze samo kopira na novu putanju kako bi se kreirao novi projekt. Ta putanja se prosljeđuje u „*ManageDB“* kako bi se otvorila konekcija na tu bazu. Pomoću metode „*ManageDB.CreateNewProject()*“ projektu se kreira Id koji se zatim odmah i ažurira. Nakon toga slijedi postavljanje početne vrijednosti vrste projekta i postavljanje aktivne zone.

```
        [HttpGet()]
        [System.Web.Http.Route("api/new-project")]
        public IHttpActionResult CreateNewProject()
        {
            var md = ManageDocs.Instance();
            try
            {
                CreateNewProjectAction.CreateNewProject();
```

```
                return Ok("Created new project!");
            }
            catch (Exception ex)
            {
                return BadRequest(ex.Message);
            }
        }
```

Isječak koda 13. Prikaz kontrolera za kreiranje projekta [autorski rad]

Kontroler za kreiranje projekta ne prima nikakve argumente od strane klijenta. Na klijentu je samo da napravi poziv na putanju „/api/new-project“, a kod u servisu za kreiranje novog projekta će u pozadini riješiti svu potrebnu logiku.

Sljedeća migrirana metoda je metoda za ažuriranje vrijednosti odabranog projekta. Prikaz metode prikazan je na isječku koda 14.

```
     public class Props
    {
        public string Key { get; set; }
        public string Value { get; set; }
        public string Name { get; set; }
        public int? Id { get; set; }
    }
    public class UpdateFieldAction
    {
        public static void UpdateValue(Props props)
        {
            int projectId = ManageDB.GetProjectID();

            ManageDB.ExecuteNonQuery($"UPDATE Projekt SET {props.Key}='" +
props.Value + "' WHERE Id='" + projectId + "';");

            System.GC.Collect();
            System.GC.WaitForPendingFinalizers();

        }
    }
```

Isječak koda 14. Prikaz metode UpdateValue [autorski rad]

Kao što je moguće vidjeti metoda kao parametar prima objekt koji se sastoji od sljedećih svojstva:

- Key – predstavlja naziv svojstva kojeg se uređuje (npr. NazivProjekta)
- Value – predstavlja novu vrijednost unesenu od strane klijenta

- Id – to je opcionalno svojstvo koje može biti korišteno kao ID pri „WHERE Id“ dijelu upita

Ova metoda direktno koristi metode nad bazom podataka što bi se kod migracija pod svaku cijenu trebalo izbjegavati, ali ovo je bio jedini način kako bi se ova funkcionalnost mogla implementirati. Pošto su u originalnom projektu korištene metode koje sadrže elemente koji su strogo vezani uz WindowsForme, a takve metode ne mogu se koristiti na serveru iz razloga jer će doći do NullExceptiona, jer ti elementi forma ne postoje.

```
[System.Web.Http.HttpPut()]
[System.Web.Http.Route("api/update-value")]
public IHttpActionResult SaveProject([FromBody]Props props)
{
    try
    {
        UpdateFieldAction.UpdateValue(props);
        return Ok("Uspješno ažuriran podatak!");
    }
    catch (Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

Isječak koda 15. Prikaz kontrolera za ažuriranje vrijednosti [autorski rad]

Kontroler sa klijenta prima objekt koji je opisan u prethodnom odlomku. Taj objekt se sa klijenta ne šalje kao dio putanje kako se dosad izvršavala komunikacija sa klijenta to servera nego se taj objekt šalje u tijelu zahtjeva. Zbog toga nam je potreban „[FromBody]“ pokraj props parametra. Taj se objekt zatim šalje u servis u kojem se izvršava sva potrebna logika.

Sljedeća migrirana metoda je metoda za dodavanje zone u odabrani projekt. Prikaz metode prikazan je na isječku koda 16.

```
public class AddZoneAction
{
    public static void AddZone()
    {
        ZonaCollection.Instance.Add();
    }
}
```

Isječak koda 16. Prikaz kontrolera za dodavanje zone [autorski rad]

Ovaj servis je vrlo jednostavan, u odnosu na prethodne servise nije bilo potrebno izdvajati iz postojećih metoda kako bi sve proradilo. Ovdje je bilo potrebno samo pozvati „*Add*“ metodu na instanci kolekcije zona. Postojeća poslovna logika obavi sve potrebno u pozadini.

```
[HttpGet()]
[System.Web.Http.Route("api/add-zone")]
public IHttpActionResult AddZone()
{
    try
    {
        AddZoneAction.AddZone();
        return Ok("Zona uspješno dodana!");
    }
    catch (Exception error)
    {
        return BadRequest(error.Message);
    }
}
```

Isječak koda 17. Prikaz kontrolera za dodavanje zone [autorski rad]

Kontroler u ovom slučaju također ne prima nikakve parametre od strane klijenta. Podaci o projektu dostupni su programu od kada korisnik otvara projekt po Id-u. Potrebno je samo da korisnik napravi poziv prema „*api/add-zone*“ te će servis i sva logika koja se vrti u pozadini dodati zonu u pravi projekt.

Sljedeća migrirana metoda je metoda za uklanjanje zone u odabrani projekt. Prikaz metode prikazan je na isječku koda 18.

```
public class DeleteZoneAction
{
    public static void DeleteZone(int id)
    {
        var zone = ZonaCollection.Instance.Find(id);
        ZonaCollection.Instance.Remove(zone);
    }
}
```

Isječak koda 18. Prikaz metode DeleteZone [autorski rad]

Metoda „*DeleteZone*“ od upravljača kao parametar prima Id zone. Taj Id zone koristi se kako bi se u kolekciji zona pronašla zona za koju je korisnik poslao Id. Zatim kad se ta zona pronađe nad kolekcijom se izvršava metoda „*Remove*“ koja zatim uklanja zonu iz kolekcije.

```
[HttpDelete()]
[System.Web.Http.Route("api/delete-zone/{id}")]
public IHttpActionResult DeleteZone(int id)
{
    try
    {
       DeleteZoneAction.DeleteZone(id);
       return Ok("Zona uspješno uklonjena!");
    }
    catch (Exception error)
    {
        return BadRequest(error.Message);
    }
}
```

Isječak koda 19. Prikaz kontrolera za brisanje zone [autorski rad]

U kontroleru za brisanje zone Id zone šalje se kao dio putanje koju korisnik poziva sa klijentske strane. To je moguće vidjeti u „Route“ dijelu koda gdje se na standardnu putanju do upravljača u vitičastim zagradama još dodaje taj „id“ dio. Zatim se taj Id prosljeđuje do servisa u kojem se obradi sva potrebna logika.

Sljedeća migrirana metoda je metoda za ažuriranje vrijednosti odabrane zone. Prikaz metode prikazan je na isječku koda 20.

```
public class UpdateZoneFieldAction
{
    public static void UpdateValue(Props props)
    {
        ManageDB.ExecuteNonQuery($"UPDATE Zone SET {props.Key}='" + props.Value
+ "' WHERE Id='" + props.Id + "';");

        System.GC.Collect();
        System.GC.WaitForPendingFinalizers();
    }
}
```

Isječak koda 20. Prikaz metode za ažuriranje vrijednosti zone [autorski rad]

Princip rada ove metode isti je kao i kod ažuriranja vrijednosti pri projektu, props je istog tipa kao i kod projekta gdje je opisano što koji parametar znači. U ovom slučaju ovaj Id predstavlja Id zone koju je potrebno ažurirati.

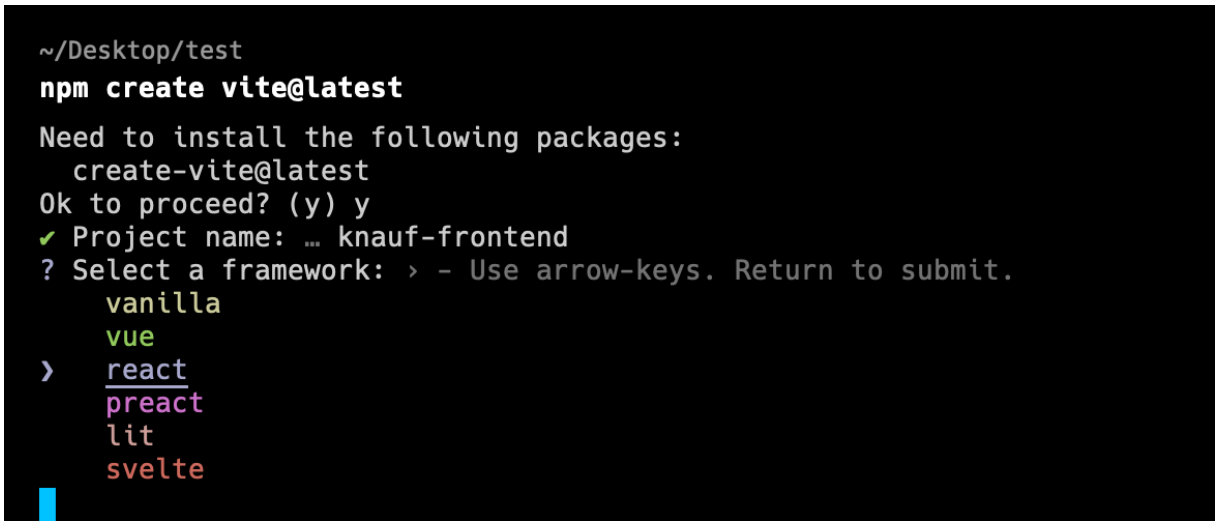
```
[System.Web.Http.HttpPut()]
[System.Web.Http.Route("api/update-zone-value")]
public IHttpActionResult SaveZone([FromBody] Props props)
{
    try
    {
        UpdateZoneFieldAction.UpdateValue(props);
        return Ok("Uspješno ažuriran podatak!");
    }
    catch (Exception ex)
    {
        return BadRequest(ex.Message);
    }
}
```

Isječak koda 21. Prikaz kontrolera za ažuriranje vrijednosti zone [autorski rad]

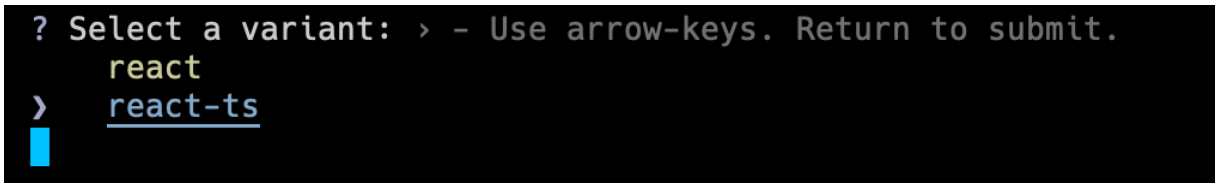
Princip rada kontrolera za ažuriranje vrijednosti zone funkcionira na isti način kao i kod ažuriranja vrijednosti projekta. Sa klijenta u tijelu zahtjeva dolazi objekt kojeg klijent šalje iz aplikacije, zatim se taj objekt šalje u servis gdje se zatim izvrši ažuriranje samog projekta.

5.3.3. Migracija korisničkog sučelja

Kao što je već navedeno u planiranju migracije korisničko sučelje bilo je potrebno izraditi ispočetka. Novo korisničko sučelje nije slijedilo dizajn već postojećeg sučelja nego je redizajnirano da bude čim jednostavnije za korisnika te da više poprimi izgled neke web aplikacije. Kako bi se kreirala nova aplikacija potrebno je pratiti sljedeće korake.



Slika 10. Proces instalacije tehnologije za izradu korisničkog sučelja 1.dio [autorski rad]

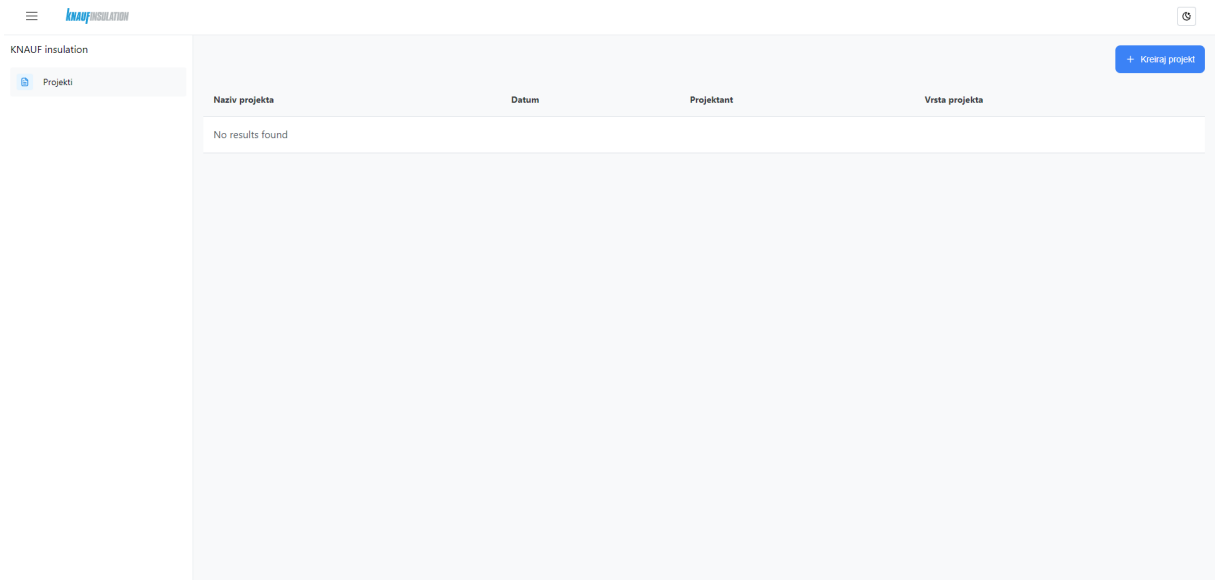


Slika 11. Proces instalacije tehnologije za izradu korisničkog sučelja 2.dio [autorski rad]

Slika 12. Proces instalacije tehnologije za izradu korisničkog sučelja 3.dio [autorski rad]

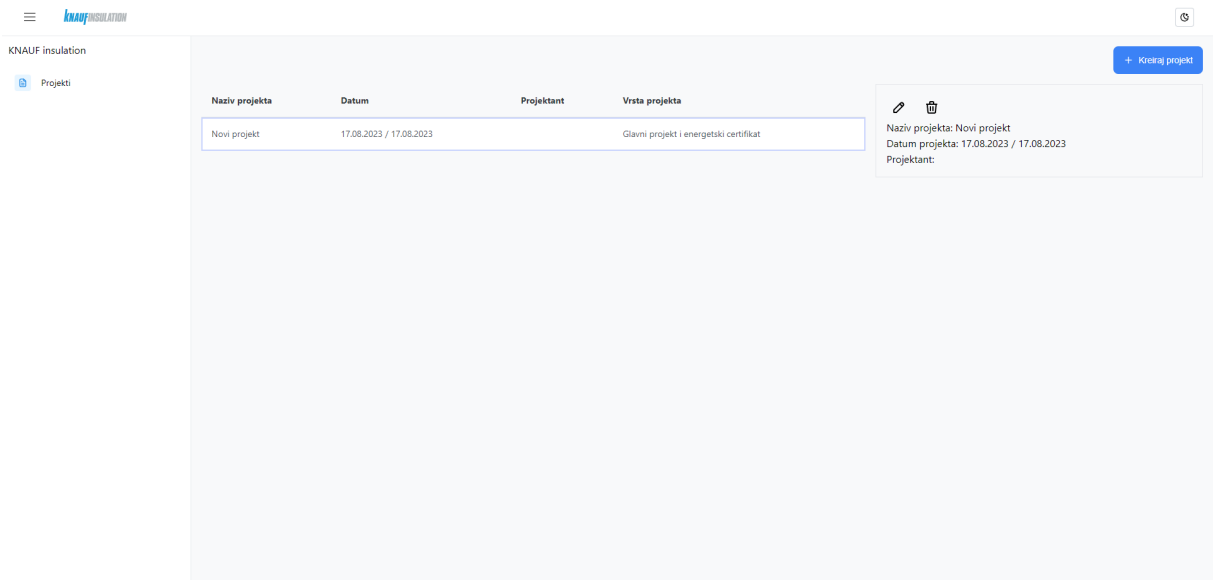
U slikama 10, 11 i 12 prikazan je proces instalacije tehnologije za izradu korisničkih sučelja. Na slici 10 vidljivo je kako je naredba ***npm create vite@latest*** početna naredba koja pokreće proces instalacije. Zatim treba unijeti ime projekta. Nakon toga potrebno je odabrati tehnologiju u kojoj će aplikacija biti izrađena. U našem slučaju to je bio React i na slici 11. vidljivo je kako je odabrana ***react-ts*** verzija. To jest React tehnologija sa TypeScript programskim jezikom. Za kraj slika 12 prikazuje završne korake kako instalirati sve potrebne module i kako zatim pokrenuti aplikaciju.

Kao već spomenuto aplikacija nije napravljena po uzoru na trenutnu aplikaciju. Napravljena je u drugačijem dizajnu na način da se korisniku fokus svede na najbitnije dijelove. Početna stranica prikazuje običnu tablicu koja pokazuje popis projekata. Kao što je vidljivo na slici 13.



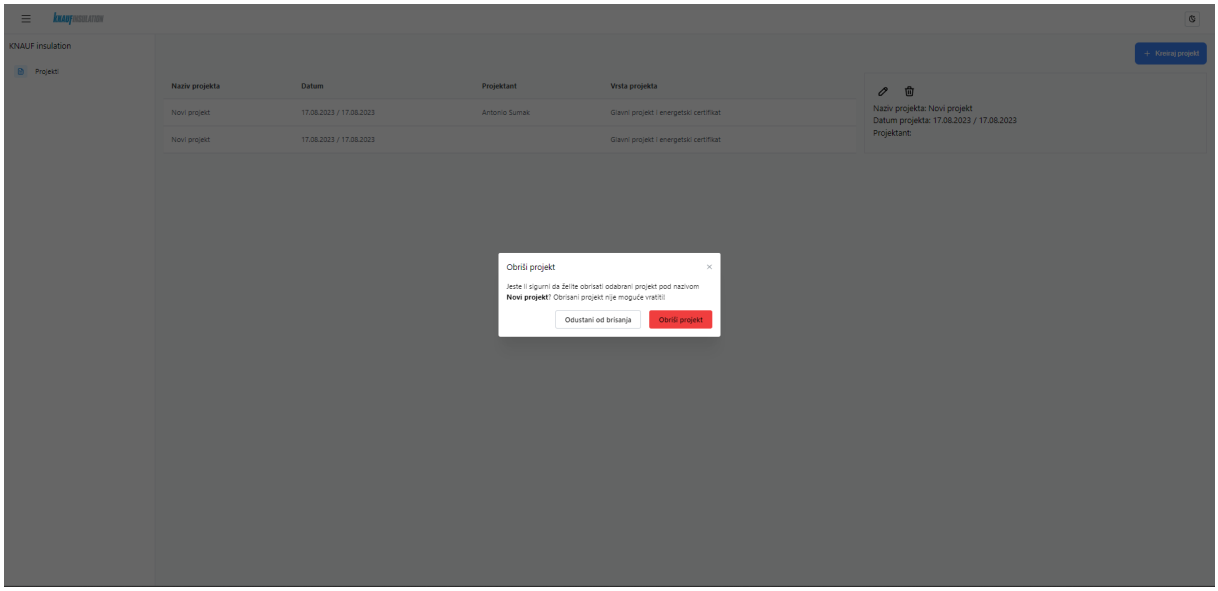
Slika 13. Početni prikaz novog korisničkog sučelja [autorski rad]

Početni zaslon web aplikacije sastoji se od tablice i gumba. Sa strane postoji navigacijska traka koja trenutno ima samo karticu „Projekti“, ali sa daljnjim razvojem aplikacije tu navigacijsku traku mogli bi popuniti sa dodatnim funkcionalnostima i stranicama. U centralnom dijelu aplikacije nalazi se prozor koji kao već spomenuto ima tablicu za prikaz podataka o projektu. Gumb nam služi kako bi se kreirao projekt. Klik na projekt u tablici također otvara dodatan dio na zaslonu koji otkriva detalje projekta, te mogućnost brisanja ili uređivanja projekta. Prikazano na sljedećoj slici.



Slika 14. Prikaz detalja o projektu [autorski rad]

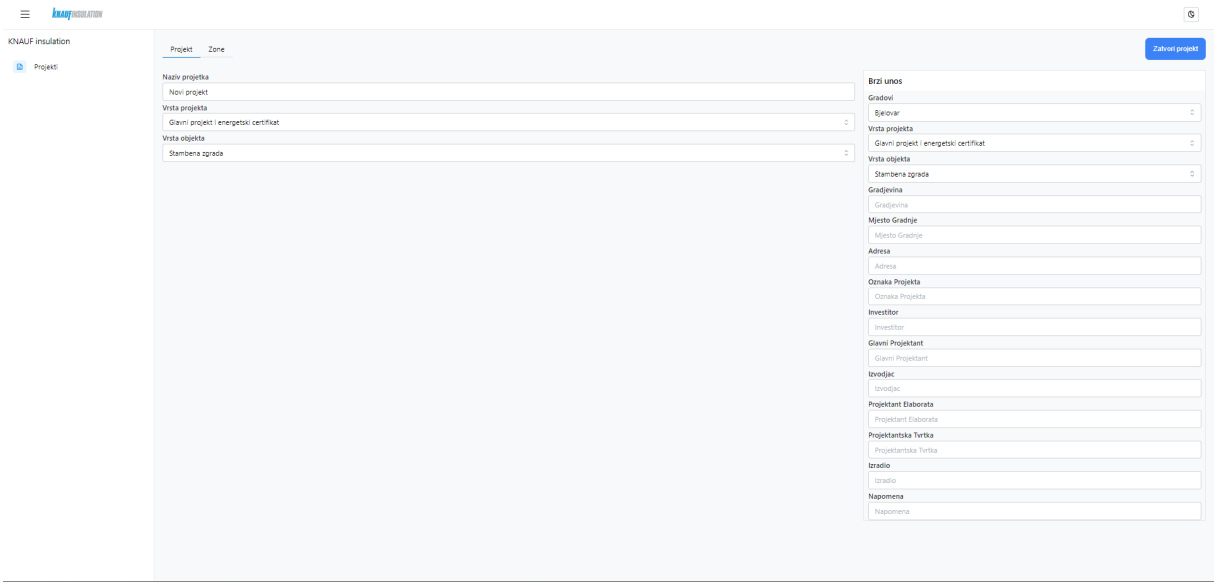
Klikom na projekt u tablici otvara se dodatan prozor koji se sastoji od nekih osnovnih detalja o projektu. Također u gornjem lijevom kutu nalaze se ikonice za brisanje i uređivanje projekta. Klik na ikonicu vodi na posebnu stranicu koja sadrži sve detalje o projektu, dok ikonica za brisanje otvara dijalog putem kojeg je moguće obrisati projekt.



Slika 15. Prikaz dijaloga za brisanje projekta [autorski rad]

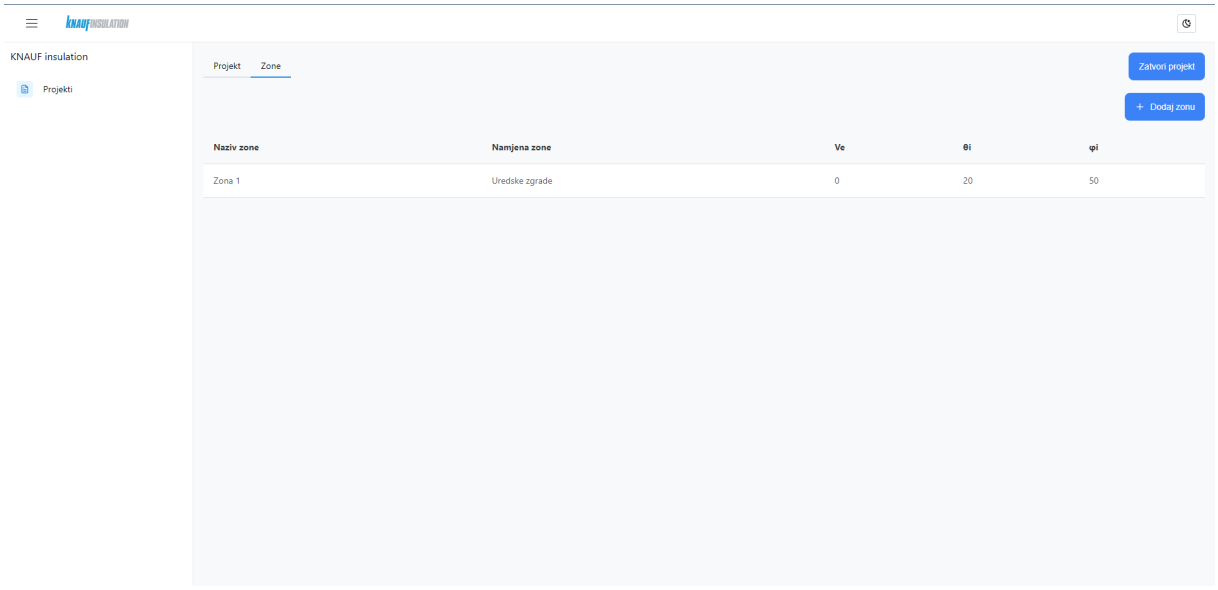
Klikom na ikonicu za brisanje projekta otvara se dijalog u kojem se korisniku ispisuje poruka ako je siguran da želi obrisati projekt. Ako klikne na „Odustani od svega“ modal se zatvara, a ako odabere „Obriši projekt“ sa klijenta se šalje HTTP zahtjev na putanju *„/api/delete-*

project/naziv_projekta“, ta putanja vodi na kontroler za brisanje projekta opisan u gornjem poglavlju. Nakon šta klijent sa servera dobije potvrdu da je projekt obrisan, opet se šalje zahtjev na server kako bi se dohvatili projekti kako bi lista bila osvježena.



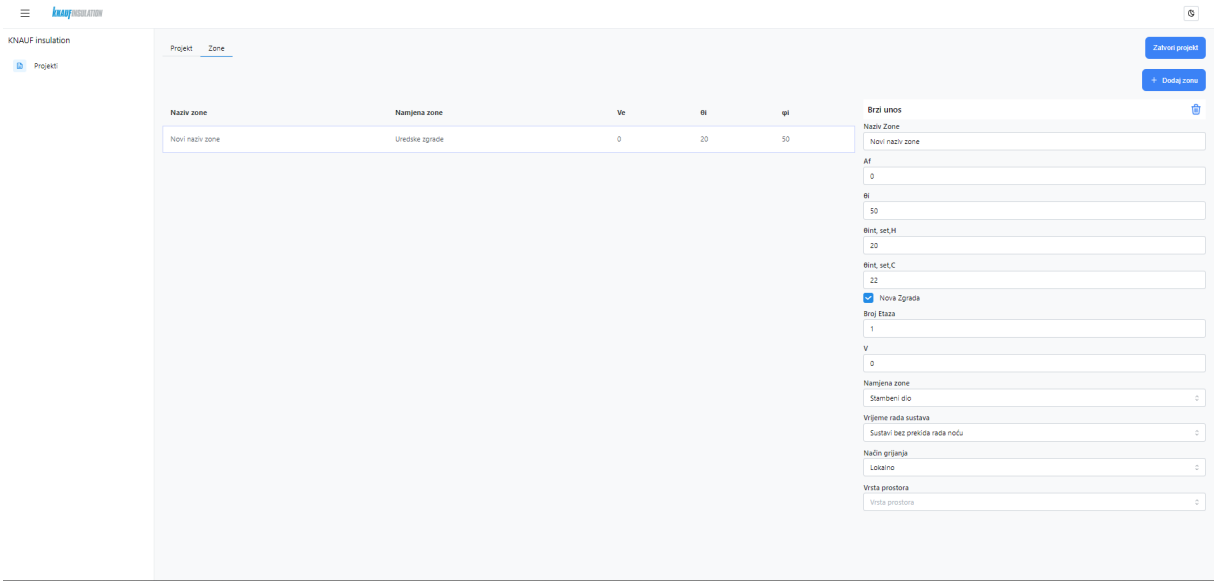
Slika 16. Prikaz detalja projekta i uređivanja projekta [autorski rad]

Klikom na ikonicu za uređivanje projekta otvara nam se nova stranica koja se sastoji od nekoliko dijelova. Na vrhu stranice imamo dvije kartice za odabir. Kartice „Projekt“ i „Zone“. Kada je odabrana kartica Projekti otvoren je sljedeći prikaz. Prikaz na kojem korisnik može mijenjati sve potrebne informacije o projektu. Kako bi se vrijednost ažurirala potrebno je samo u bilo koje polje ili na novo napisati ili urediti već postojeće vrijednosti. Nakon što korisnik upiše vrijednost ta vrijednost se u obliku objekta šalje HTTP zahtjev na putanju „/api/update-value“, zatim se na serveru izvrši sva potrebna logika vezana uz upise u bazu podataka.



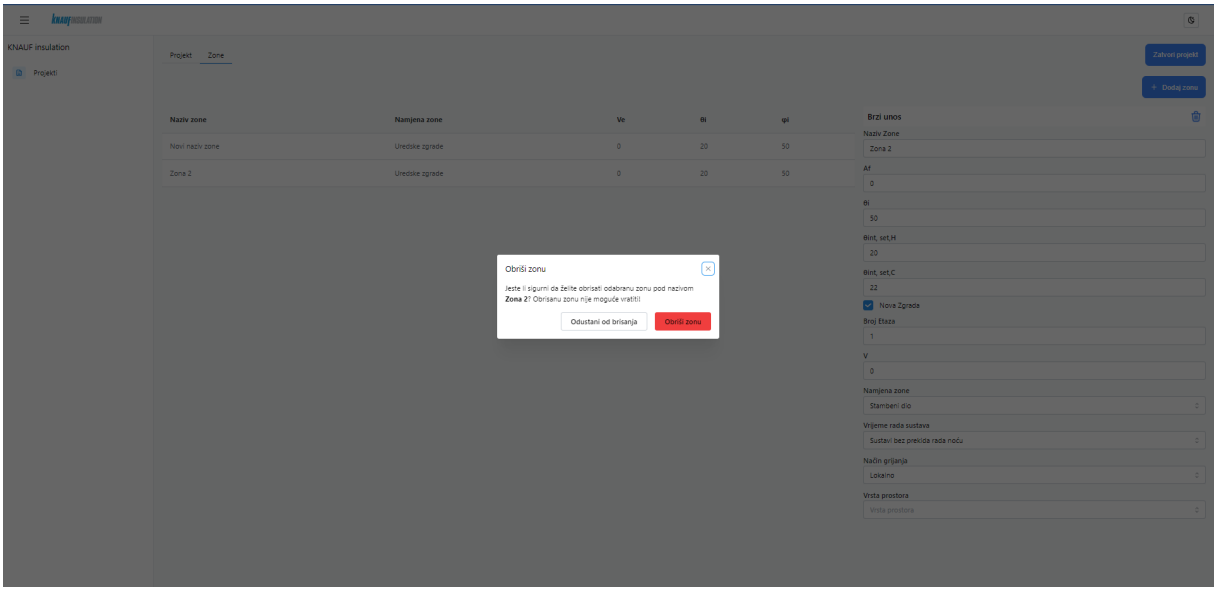
Slika 17. Prikaz tablice zona [autorski rad]

Kada odaberemo karticu „Zone“ dobivamo sljedeći prikaz. Slično kao i sa prikazom tablice projekta, u tablici za zone moguće je vidjeti osnovne podatke, a isto tako klikom na određeni red u tablici otvara se dodatan prozor sa detaljnim prikazom zona te mogućnost brisanja i uređivanja zone.



Slika 18. Prikaz detalja zone [autorski rad]

Klikom na red u tablici otvara se prozor sa detaljima o zoni. Kako bi se ažurirala vrijednost zone potrebno je isto kao i kod projekta upisati vrijednost i zatim se vrši poziv na *„/api/update-zone-value“* gdje se na kontroler šalje objekt koji je isti kao i kod projekta, samo sad se zajedno sa ostalim vrijednostima šalje i Id zone.



Slika 19. Prikaz dijaloga za brisanje zone [autorski rad]

Isto kao i kod projekta, na pritisak na ikonicu za brisanje zone otvara se dijalog za brisanje zone. Ako se klikne na „Odustani od brisanja“ dijalog se zatvara, a ako se šalje „Obriši zonu“ na putanju „/api/delete-zone“ te se još dodatno u putanju dodaje Id zone koju se želi obrisati.

```
import axios from 'axios';
import { IProject } from '../../../core/Types/ProjectTypes';
import { API_URL } from '../../../core/Services/Constants';

class ProjectService {
  async getAllProjects(): Promise<IProject[]> {
    const response = await axios.get(`${API_URL}/api/get-projects`);

    response.data.forEach((project: IProject) => {
      project.Id = project.Putanja.split('\\').at(-1)?.split('.').at(0);
    });
    return response.data;
  }

  async getProjectById(id: string): Promise<any> {
    const response = await axios.get(`${API_URL}/api/project/${id}`);

    return response.data;
  }

  async createNewProject(): Promise<any> {
    return await axios.get(`${API_URL}/api/new-project`);
  }

  async updateValue(props: {
    Key: string;
    Value: string | number;
    Name: string;
  }) {
    return await axios.put(`${API_URL}/api/update-value`, props);
  }

  async deleteProject(id: string) {
    return await axios.delete(`${API_URL}/api/delete-project/${id}`);
  }

  async closeProject() {
    return await axios.get(`${API_URL}/api/close-project`);
  }
}

export default new ProjectService();
```

Isječak koda 22. Servis za projekte [autorski rad]

```
import axios from 'axios';
import { API_URL } from '../../../core/Services/Constants';
class ZoneService {
  async addNewZone() {
    return await axios.get(`${API_URL}/api/add-zone`);
  }

  async deleteZone(id: number) {
    return await axios.delete(`${API_URL}/api/delete-zone/${id}`);
  }

  async updateValue(props: {
    Key: string;
    Value: string | number;
    Name?: string;
    Id: number;
  }) {
    return await axios.put(`${API_URL}/api/update-zone-value`, props);
  }
}

export default new ZoneService();
```

Isječak koda 23. Servis za zone [autorski rad]

Slika 20 i 21 prikazuju kako izgledaju servisi koji povezuju kontrolere na serveru sa klijentom na klijentskoj strani. To su sve metode koje izvršavaju HTTP zahtjeve GET, POST, PUT i DELETE tipa prema serveru. Kako bi se na serveru izvršilo nešto potrebno je da klijent to zatraži od svoje strane nekom akcijom (pritisak gumba, promjena vrijednosti…). Tim akcijama pozivaju se ove metode koje zatim preko HTTP zahtjeva pozivaju sa ili šalju na server.

```
export function GenerateInputs(data: any) {
  const inputs: {
    id: string;
    label: string;
    type: string;
    value: number | string;
  }[] = [];
  for (const key in data) {
    if (key.includes('str')) continue;
    if (skipValues[key]) continue;
    const label = key.split(/(?=[A-Z])/).join(' ');

    inputs.push({
      label: label.includes('Id') ? label.replace('Id', '') : label,
      type: label.includes('Id') ? 'select' : 'text',
```

```
      value: data[key],
      id: key,
    });
  }

  return inputs;
}
```

Isječak koda 24. Generiranje konfiguracije za kreiranje UI elementa [autorski rad]

Kako aplikacija koja se migrira ima automatsko generiranje polja za unos cilj je bio da se na neki način i to preslika na novu aplikaciju. Ova metoda GenerateInputs iz podataka koje dobiva sa servera generira konfiguraciju koja se kasnije koristi kod pravljenja korisničkog sučelja. Neovisno o strukturi podataka koje dobije, prema tipu tih podataka kreira određenu konfiguraciju koja sadrži:

- Label – Koristi se za prikaz imena svojstva u label komponenti
- Type – Primarno za provjeru da li je vrijednost textualnog oblika
- Value – vrijednost koja dolazi sa servera za to svojstvo
- Id – koristi se kao ključ za komponentu

```
{project &&
    GenerateInputs(project).map((value, idx) =>
        <Input.Wrapper label={value.label} key={idx}>
            <Input
                placeholder={value.label}
                defaultValue={value.value}
                onBlur={async (e) =>
                    await ProjectService.updateValue({
                        Key: value.id,
                        Value: e.target.value,
                        Name: projectId ?? '',
                    })
                }
            />
        </Input.Wrapper>
    )}
```

Isječak koda 25. Primjena GenerateInputs metode u komponenti [autorski rad]

Vrijednost koju GenerateInputs vraća je polje. Uz pomoć map funkcije prolazimo kroz te vrijednosti unutar polja te za svaku vrijednost u tom polju generiramo polje unosa zajedno sa pripadajućom labelom. Ovdje je i moguće vidjeti kako se ažuriraju podaci nakon unosa ili

promjene podataka. Nakon što se makne fokus sa unosnog polja pokreće se updateValue metoda koja šalje potreban objekt na server.

```
export interface IProps {
  data: any[];
  headers: Record<string, { header: string; visible: boolean }>;
  setRowData: any;
}

export default function DynamicTable({ data, headers, setRowData }: IProps) {
  return (
    <DataTable
      value={data}
      selection="single"
      onSelectionChange={(e: any) => setRowData(e.value)}
      stateStorage="session"
      stateKey="dt-state-demo-local"
      dataKey="id"
      selectionMode="single"
    >
      {Object.keys(headers).map((key) => (
        <Column
          className="text-sm"
          key={key}
          field={key}
          header={headers[key].header}
        />
      ))}
    </DataTable>
  );
}
```

Isječak koda 26. Generička tablica [autorski rad]

Tablice koje se koriste za prikaz projekata i zona stvorene su iz iste komponente. Komponenta koristi headers parametar kao glavni generator stupaca tablice. Npr. kako bi se kreirala tablica za projekte potrebno je da headers ima sljedeći oblik:

```
  const tableHeaders = {
    NazivProjekta: { header: 'Naziv projekta', visible: true },
    Datum: { header: 'Datum', visible: true },
    Projektant: { header: 'Projektant', visible: true },
    VrstaProjekta: { header: 'Vrsta projekta', visible: true },
  };
```

Isječak koda 27. Oblik konfiguracije za tablicu [autorski rad]

Zatim je potrebno iskoristitit tu samu komponentu, tablica kao komponenta koristi se na sljedeći način:

```
<DynamicTable
              headers={tableHeaders}
              data={projects}
              setRowData={handleChange}
            />
```

Isječak koda 28. Korištenje tablice [autorski rad]

Tablica se poziva kao komponenta i zatim se kao parametri prosljeđuju zaglavlje za tablice u gore navedenom obliku, zatim podaci za tablicu i na kraju imamo seter koji na klik na red tablice postavlja podatke toga odabranog reda u varijablu.

5.4. Generički generator

Zbog kompleksnosti postojećeg rješenja, na posebnom projektu prikazano je kako bi se postojeći objekti kao što su npr. zona, postojeći projekt, građevinski dijelovi itd. mogli prikazivati na sučelju klijenta bez prevelikih promjena na postojećem rješenju. Zamišljeno je da se iz atributa svojstva kreira JSON objekt koji se zatim šalje na klijent gdje se iz toga JSON objekta generira korisničko sučelje za unos vrijednosti od strane korisnika.

5.4.1. Backend generator

Kako bi generator uopće mogao generirati JSON prvo je potrebno pravilno kreirati atribute klase. Pravilno kreiran atribut trebao bi izgledati na sljedeći način.

```
[DisplayName("Naziv zgrade")]
        [System.ComponentModel.DescriptionAttribute("Naziv građevine. Podatak se
koristi za energetski certifikat.")]
        [DefaultValue("")]
        [BrowsableAttribute(true)]
        [ReadOnlyAttribute(false)]
        public string NazivZgrade
        {
            get
```

```
            {
                return _nazivZgrade;
            }
            set
            {
                _nazivZgrade = value;
            }
        }
```

Isječak koda 29. Pravilan izgled atributa [autorski rad]

Atributi koji su nam potrebni su:

- DisplayName
- DescriptionAttribute
- DefaultValue
- BrowsableAttribute
- ReadOnlyAttribute

Iz ovih atributa generira se JSON koji se zatim šalje na klijenta. Kako bi se JSON kreirao potrebna nam je metoda koja sve te atribute pretvara u JSON.

```
    public static class Generator
    {
        public static dynamic ConvertToJsonPropertyInfo(object obj)
        {
            var propertyInfoList = new List<object>();

            var properties = obj.GetType().GetProperties();
            foreach (var property in properties)
            {
                var displayName =
property.GetCustomAttributes(typeof(DisplayNameAttribute), true)
                    .OfType<DisplayNameAttribute>()
                    .FirstOrDefault()?.DisplayName;

                var description =
property.GetCustomAttributes(typeof(DescriptionAttribute), true)
                    .OfType<DescriptionAttribute>()
                    .FirstOrDefault()?.Description;

                var propertyType = property.PropertyType.Name;

                var value = property.GetValue(obj);

                var isBrowseable =
property.GetCustomAttributes(typeof(BrowsableAttribute), true)
```

```
                    .OfType<BrowsableAttribute>()
                    .FirstOrDefault()?.Browsable;

                var isReadOnly =
property.GetCustomAttributes(typeof(ReadOnlyAttribute), true)
                    .OfType<ReadOnlyAttribute>()
                    .FirstOrDefault()?.IsReadOnly;

                propertyInfoList.Add(new
                {
                    DisplayName = displayName,
                    Description = description,
                    Type = propertyType,
                    Value = value,
                    IsBrowseable = isBrowseable,
                    IsReadOnly = isReadOnly,
                });
            }

            return JsonConvert.SerializeObject(propertyInfoList,
Formatting.Indented);
        }
    }
```

Isječak koda 30. Izgled Generator metode [autorski rad]

U generator metodi sa objekta kojeg proslijedimo u metodu dobivamo sve atribute uz pomoć metode „.GetProperties()“. Zatim prolazimo kroz sve te atribute i iz njih uzimamo potrebne vrijednosti koje zatim dodajemo u „propertyInfoList“ listu. Tu listu zatim pretvaramo u JSON objekt.

```
        public string Get()
        {
            var zone = new Zona();
            string json = Generator.Generator.ConvertToJsonPropertyInfo(zone);

            return json;
        }
```

Isječak koda 31. Pozivanje generator metode [autorski rad]

Prvi korak je instanciranje željene klase. Zatim taj objekt šaljemo u generator metodu koja zatim vraća JSON oblik tog objekta, zatim taj objekt šaljemo na klijenta kad klijent napravi zahtjev prema serveru.

Server generira JSON oblik prikazan na sljedećoj slici. U takvom obliku šalje se na klijentsku aplikaciju.

```
  {
    "DisplayName": "Naziv zgrade",
    "Description": "Naziv građevine. Podatak se koristi za energetski certifikat.",
    "Type": "String",
    "Value": "",
    "IsBrowseable": true,
    "IsReadOnly": true
  },
  {
    "DisplayName": "Katastarska čestica",
    "Description": "Na kojoj katastarskoj čestici se nalazi zgrada.",
    "Type": "String",
    "Value": "",
    "IsBrowseable": true,
    "IsReadOnly": false
  },
  {
    "DisplayName": "Nova zgrada",
    "Description": "Ako je riječ o novoj zgradi vrijednost je istinita.",
    "Type": "Boolean",
    "Value": true,
    "IsBrowseable": true,
    "IsReadOnly": false
  },
  {
    "DisplayName": "Lokacija zgrade",
    "Description": "Lokacija zgrade. Podatak se koristi u energetskom
certifikatu.",
    "Type": "String",
    "Value": "",
    "IsBrowseable": true,
    "IsReadOnly": false
  },
  {
    "DisplayName": "Gradovi",
    "Description": "Moguća lista gradova",
    "Type": "List`1",
    "Value": ["Zagreb", "Varaždin", "Osijek"],
    "IsBrowseable": true,
    "IsReadOnly": false
  }
]
```

Isječak koda 32. Izgled konfiguracije za kreiranje UI-a [autorski rad]

5.4.2. Frontend generator

Kao što je moguće vidjeti u isječku koda 32. JSON nam predstavlja listu elemenata koje treba kreirati na klijentu. Objekt se sastoji od „DisplayName“ koji predstavlja vrijednost koja će se prikazivati u labeli polja za unos, zatim imamo „Description“ koji detaljnije opisuje taj atribut. Zatim „Type“ koji nam služi za kreiranje tipova polja za unos, ako je tip „String“ ili „Int8“ ili „int16“ itd. kreirati će se standardno polje za unos u kojeg korisnik samostalno upisuje vrijednosti. Ako je „Type“ vrijednosti Boolean onda umjesto standardnog polja za unos prikazati će se CheckBox unos, a ako je „Type“ vrijednosti „List“ korisniku će se prikazati polje u kojem korisnik odabire iz ponuđenih vrijednosti.

```
export default function GenerateInputs({
  inputConfig,
}: {
  inputConfig: IConfig[] | null;
}) {
  const numberNaming = ['Int32', 'Single', 'Int16', 'Int8'];
  return inputConfig?.map((input, index) => {
    console.log(input.Type);
    if (numberNaming.includes(input.Type)) {
      return (
        <Input.Wrapper key={index * Math.random()} label={input.DisplayName}>
          <Input type='number' disabled={input.IsReadOnly === true} />
        </Input.Wrapper>
      );
    }
    if (input.Type === 'String') {
      return (
        <Input.Wrapper key={index * Math.random()} label={input.DisplayName}>
          <Input type='string' disabled={input.IsReadOnly === true} />
        </Input.Wrapper>
      );
    }
    if (input.Type === 'Boolean') {
      return (
        <Checkbox
          label={input.DisplayName}
          value={input.Value}
          checked={input.Value}
          disabled={input.IsReadOnly === true}
        />
      );
    }
    if (input.Type.includes('List')) {
      return (
        <Select
          data={input.Value}
          label={input.DisplayName}
```

```
          searchable={input.IsBrowseable ?? false}
          disabled={input.IsReadOnly === true}
        ></Select>
      );
    }
  });
}
```

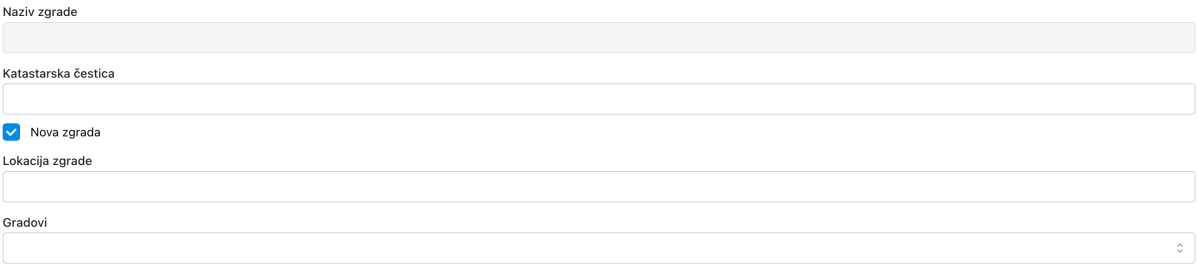
Isječak koda 33. Frontend komponenta generiranja sučelja [autorski rad]

U ovoj komponenti je moguće vidjeti na koji način se prikazuje koje polje za unos. Prema tipu atributa generiraju se potrebna polja za unos. Jedino što je potrebno da ova komponenta kreira to korisničko sučelje je taj JSON koji dolazi sa servera, a pozivanje komponente izgleda ovako.

```
<div className='wrapper'>
  <GenerateInputs inputConfig={config} />
</div>
```

Isječak koda 34. Pozivanje komponente [autorski rad]

Potrebno je samo pozvati komponentu „GenerateInputs“ i kao parametar joj proslijediti „inputConfig“ što predstavlja JSON koji dolazi sa servera.



Slika 20. Izgled generiranog zaslona [autorski rad]

Na slici prikazano je kako izgleda zaslon koji je generiran iz gore navedenog JSON oblika.

5.5. Nastavak migracije

Kako bi se proces migracije nastavio potrebno je da se postojeći kod dobro razradi i po mogućnosti prikaže u grafičkom obliku. Taj korak olakšao bi sam proces migracije. Trenutno bez takve grafičke dokumentacije potrebno je ručno prolaziti kroz kod i tražiti elemente koji na sebi imaju „Event listenere“. Na taj način najlakše je pronaći početak izvršavanja koda za

odabranu akciju. Uz sam pronalazak početka izvršavanja, kroz kod je potrebno postavljati „breakpointeve“ koji olakšavaju razumijevanje samog protoka podataka kroz aplikaciju. Što se tiče samog koda, kad se i pronađe željena metoda koja izvršava to što nam je potrebno obično sadrži neke metode koje se koriste vezano uz samu formu. Iz tog razloga nije moguće direktno pozivanje tih metoda jer sadrže kod koji će na serveru izbacivati greške jer ne postoji instanca forme. To znači da za svaku funkcionalnost potrebno je kreirati kontroler koji služi kao pristupna točka za klijenta i servis za tu funkcionalnost u koju je ručno potrebno izdvojiti kod, to jest metode. Ako metoda ne sadrži elemente ili metode namijenjene za windows forme, tu metodu moguće je direktno pozvati, bez ikakvog odvajanja koda.

6. Zaključak

U posljednje vrijeme moguće je primijetiti kako su stolne aplikacije sve manje u upotrebi. Razlog toga su web aplikacije koje nude bolje uvjete za razvoj ali isto tako i bolje uvjete za korisnike. Web aplikacija dostupna je na bilo kojem uređaju dok su stolne aplikacije vezane striktno za računalo te u većini slučajeva vezane za određeni operacijski sustav, što znači da ih nije moguće koristiti na svim operacijskim sustavima. Smatram da ako bi klijent zahtijevao stolnu aplikaciju da se ne koriste Windows Forme ili WPF nego da se koristi Elektron koji može kreirati aplikacije koje je moguće pokrenuti na svim operacijskim sustavima, ili još bolje kreirati web aplikaciju koja je PWA tip web aplikacije. Što znači da je web aplikaciju moguće preuzeti na računalo i koristiti kao stolnu aplikaciju.

Odabir tehnologija za izradu novog korisničkog sučelja bio je također dosta opširan, ali smatram da se React pokazao kao pravilan odabir. React kao tehnologija sadržavao je sve potrebne funkcionalnosti. Te funkcionalnosti su globalne pohrana stanja aplikacije, prikazivanje različitih komponenti na različitim putanjama, jednostavnost korištenja i skalabilnost za buduće povećanje aplikacije.

Kako je ova aplikacija dosta kompleksna i potrebno je više osoblja kako bi se aplikacija u potpunosti migrirala. Uspješno su migrirane osnovne funkcionalnosti, ali ne u potpunosti. Zato je dodatno napravljen generator koji pretvara bilo koji objekt u JSON oblik. Taj generator olakšava postupak migracije iz razloga jer uzima sve atribute i iz njih pravi JSON, bez da je potrebno ručno izdvajati atribute koji bi se koristili na klijentu.

7. Popis literature

[1] S. Purewal, Learning Web App Development, Sebastopol: O'Reilly Media, 2014.

[2] J. C. P. Michael S. Mikowski, Single Page Web Applications, New York: Manning, 2014.

[3] C. Y. Joe Fender, Front-End Fundamentals, Leanpub, 2015.

[4] »17 JavaScript Frameworks that You Should Know About – A Comprehensive Guide,« Codemotion Magazine, 5 Srpanj 2021. [Mrežno]. Available: https://www.codemotion.com/magazine/frontend/javascript/javascript-frameworks-guide/. [Pokušaj pristupa 7 Srpanj 2022].

[5] J. Clark, »Top 10 Backend Technologies,« Back4App, [Mrežno]. Available: https://blog.back4app.com/backend-technologies/. [Pokušaj pristupa 7 Srpanj 2022].

[6] M. Massé, REST API, Sebastopol: O'Reilly Media, 2012.

[7] »FRONT-END FRAMEWORKS,« State of JS, 2021. [Mrežno]. Available: https://2021.stateofjs.com/en-US/libraries/front-end-frameworks/. [Pokušaj pristupa 7 Srpanj 2022].

[8] Facebook, »React documentation,« React, [Mrežno]. Available: https://reactjs.org/docs/getting-started.html. [Pokušaj pristupa 7 Srpanj 2022].

[9] »React Ecosystem Guide,« ITNext, 25 Ožujak 2018. [Mrežno]. Available: https://itnext.io/react-ecosystem-guide-4a5f85d17623. [Pokušaj pristupa 7 Srpanj 2022].

[10] A. Insignares, »React Pros and Cons: What are the Advantages and Disadvantages of ReactJS?,« Koombea, 10 Ožujak 2021. [Mrežno]. Available: https://www.koombea.com/blog/react-pros-and-cons-what-are-the-advantages-and-disadvantages-of-reactjs/. [Pokušaj pristupa 7 Srpanj 2022].

[11] K. Varkki, »The Most Popular React UI Component Libraries in 2022,« Sitepoint, 13 Prosinac 2021. [Mrežno]. Available: https://www.sitepoint.com/popular-react-ui-component-libraries/. [Pokušaj pristupa 7 Srpanj 2022].

[12] »Angular Vs React: Difference Between Angular and React,« Interviewbit, 2 Lipanj 2022. [Mrežno]. Available: https://www.interviewbit.com/blog/angular-vs-react/. [Pokušaj pristupa 7 Srpanj 2022].

[13] Google, »Angular documentation,« Google, [Mrežno]. Available: https://angular.io/guide/what-is-angular. [Pokušaj pristupa 7 Srpanj 2022].

[14] »TypeScript documentation,« Microsoft, [Mrežno]. Available: https://www.typescriptlang.org/. [Pokušaj pristupa 7 Srpanj 2022].

[15] »Vue documentation,« Vue, [Mrežno]. Available: https://vuejs.org/guide/introduction.html#single-file-components. [Pokušaj pristupa 7 Srpanj 2022].

[16] »The Good and the Bad of Vue.js Framework Programming,« Altexsoft, 11 Rujan 2019. [Mrežno]. Available: https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/. [Pokušaj pristupa 7 Srpanj 2022].

[17] »NodeJs documentation,« NodeJs, [Mrežno]. Available: https://nodejs.org/en/about/. [Pokušaj pristupa 7 Srpanj 2022].

[18] »Three Major Traps to Avoid When Building Enterprise Node.js Apps,« Section, 18 Rujan 2022. [Mrežno]. Available: https://www.section.io/blog/enterprise-nodejs-traps-to-avoid/. [Pokušaj pristupa 7 Srpanj 2022].

[19] »What is Python used for? 10 practical Python uses,« FutureLearn, [Mrežno]. Available: https://www.futurelearn.com/info/blog/what-is-python-used-for. [Pokušaj pristupa 7 Srpanj 2022].

[20] »C# documentation,« Microsoft, [Mrežno]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/. [Pokušaj pristupa 7 Srpanj 2022].

[21] »Electron documentation,« ElectronJs, [Mrežno]. Available: https://www.electronjs.org/docs/latest. [Pokušaj pristupa 7 Srpanj 2022].

[22] »Introducing the Electron.js framework, its advantages and disadvantages,« Ded9, 16 Lipanj 2021. [Mrežno]. Available: https://ded9.com/introducing-the-electron-js-framework-its-advantages-and-disadvantages/. [Pokušaj pristupa 7 Srpanj 2022].

[23] A. Joshi, »WPF vs WinForms: 19 Pros + Cons To Uncover The Differences,« RD Global, 4 Lipanj 2019. [Mrežno]. Available: https://www.rdglobalinc.com/wpf-vs-winforms-what-to-choose/. [Pokušaj pristupa 7 Srpanj 2022].

Popis slika

Popis isječka koda

Popis tablica

Dodaci

Link na repozitorij: https://github.com/zstapic/SumakZavrsni

Backend grana: Zavrsni-backend

Frontend grana: Zavrsni-frontend

Generator frontend grana: generator-ui-frontend

Generator backend grana: generator-ui-backend