Razvoj web aplikacija u ASP.NET programskom okviru

Rasinec, Karlo

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Karlo Rasinec

Razvoj web aplikacija u ASP.NET programskom okviru

ZAVRŠNI RAD

Varaždin, 2019.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Karlo Rasinec

Matični broj: 44466/15–R

Studij: Informacijski sustavi

Razvoj web aplikacija u ASP.NET programskom okviru

ZAVRŠNI RAD

Mentor:

prof. dr. sc. Dragutin Kermek

Varaždin, Rujan 2019.

Karlo Rasinec

Izjava o izvornosti

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

Autor/autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

_____

Sažetak

Ciljevi završnog rada jesu proučiti i opisati opći proces razvoja web-aplikacije u ASP.NET okruženju (modeli podataka, pogledi, kontroleri za rukovanje HTTP zahtjevima...). uvod u web-aplikacije, tehnologije i jezici za razvoj web-aplikacija, programski jezik C#, usporedba spomenutog jezika s ostalim programskim jezicima na strani poslužitelja, ASP.NET okruženje za web-aplikacije, razvojno okruženje Visual Studio, prenosivost ASP.NET web-aplikacija, opis i usporedba programskih okvira za razvoj korisničke strane web-aplikacije te razvoj web-aplikacije za vremensku prognozu na području Hrvatske.

Ključne riječi: ASP.NET, C#, web-aplikacija, MVC uzorak, HTML, CSS, JavaScript, okviri za razvoj korisniče strane aplikacije, jezici na strani servera

Sadržaj

1. Uvod

Naziv je rada „Razvoj web aplikacije u ASP.NET programskom okviru“ pa će njegov glavni fokus biti opis tog okvira i sama izrada aplikacije. Prije svega na početku rada bit će opisani glavni jezici koji se uporabljuju za razvoj web-aplikacija, a to su HTML, CSS i JavaScript koji su prisutni u gotovo svakoj web-aplikaciji. Bitno je znati kako su uopće nastali ti jezici, kako se pojavio sam interneta te čemu svaki od tih jezika služi. HTML služi za prikaz elemenata na stranici, dok CSS služi za stavljanje stilova na te elemente, što može uključivati i promjenu samog izgleda i strukture stranice. Naposljetku, JavaScript ima raznoliku primjenu, od provjera unosa korisnika pa do promjena izgleda elementa i reakcija na događaje kao što je klik mišem. Svaka web-aplikacija uostalom ima neki jezik na strani servera kojim se koristi. Neki će biti opisani u radu, a to su C#, PHP, Python i Node.js. C# je jezik koji će biti uporabljen za izradu aplikacije i sam po sebi vrlo je snažan jezik s mnogo mogućnosti. PHP je među prvim jezicima na strani klijenta i vrlo je popularan još i danas te je pun mogućnosti i ima veliki zajednicu. Tu su Python, koji je relativno jednostavan, besplatan i pristupačan, i, nakraju, Node.js koji se koristi JavaScriptom, što može biti prednost jer je isti jezik na strani servera i klijenta. Također će biti opisane neke razvojne okoline koje općenito služe tomu da bi olakšale i ubrzale proces razvoja web-aplikacije. Svaki jezik na strani servera ima neke prednosti i neke mane koje će biti proučene u radu. Nakon tih usporedbi slijedi najveći dio teoretskog dijela ovog rada, a to je sam opis ASP.NET okvira s fokusom na MVC uzorak. MVC (eng. model-view-controler, model-pogled-kontroler) uzorak je koji odmah pri kreiranju projekta već napravi solidan temelj za izradu aplikacije i ima konvencije kojima si, ako ih programer poštuje, može skratiti posao (manje konfiguracije). Model se odnosi na podatke, najčešće se rabi za spremanje podataka koji će biti prikazani. Pogled se odnosi na ono što korisnik vidi, odnosno riječ je o HTML dokumentima (naravno, unutar njega se može pisati i jQuery i C# s pomoću Razor-stroja (eng. engine)). Potom se opisuju kontroleri koji reagiraju na zahtjeve korisnika i pozivaju sve što je potrebno da bi se obradili zahtjevi (razne metode, poglede itd.). Nakon tog, opsežnog dijela slijede opis i usporedba okvira za razvoj korisničke strane koji se rabe za skraćivanje koda i bolju efikasnost obavljanja poslova na strani klijenta. Naposljetku se opisuje sama aplikacija. Njezina je tema vremenska prognoza i glavne se funkcionalnosti sastoje od dohvata jednodnevne ili peterodnevne prognoze, s prikazom u obliku grafa i tablice, te karte na kojoj korisnik može vidjeti trenutačno vrijeme u većim hrvatskim gradovima, a zadnja je mogućnost prikaza osnovne statistike za neke gradove u Hrvatskoj (minimum, maksimum i prosjek). Sam razvoj web-

aplikacija znatno se mijenjao kroz vrijeme, jer sve je u početku bilo statično i osnovno (tekst, slike), a danas već postoje razne animacije, promjenjiv sadržaj, iscrtavanja oblika, grafova itd. Tehnologija se isto tako znatno promijenila u smislu da danas postoje okviri koji praktički mogu napraviti web-aplikaciju s vrlo malo kodiranja. U nastavku slijedi teorijski dio rada.

2. Tehnologije i jezici za razvoj web-aplikacije

U nastavku će biti opisani jezici za razvoj web aplikacija (njihova povijest i općenito nešto o tim jezicima). Nakon toga će biti opisane neke razvojne okoline.

2.1. HTML

HTML (Hyper Text Markup Language) jest jezik koji služi za izradu HTML dokumenata koje današnji preglednici mogu prikazati (jedan je dokument jedna web-stranica). Jezik je statičan, odnosno sam po sebi ne sadržava mogućnosti za animaciju ili nekakvu interaktivnu web-stranicu, naravno elementi kao `<a>` element omogućuju šetanje po dokumentu i među dokumentima, odnosno web-stranicama. U kombinaciji s drugim jezicima HTML može postati dinamičan (u smislu da se pomoću drugih jezika ostvaruje ta dinamičnost), a neki primjeri onoga što stranicu čini dinamičnom jest animacija određenih njezinih dijelova, mogućnosti korisničkog unosa (uporaba korisničkog unosa da bi se mijenjao sadržaj) i prilagođivanje sadržaja korisnikovim potrebama te, naravno, dinamički sadržaj koji se mijenja kako bi podatci na stranici uvijek bili ažurni i relevantni. Da bi se postigla ta dinamičnost, HTML se kombinira s drugim web-tehnologijama, a te su tehnologije: CSS i JavaScript, ali, da bi dinamičnost bila moguća, mora postojati i još neki jezik koji služi kao jezik na strani servera koji omogućuje dinamično dohvaćanje sadržaja [1].

2.1.1. Povijest

Tim Berners-Lee kreator je HTML jezika čiji je razvoj započeo 1990. godine, a 1992. godine u uporabi su već bili osnovni elementi kao što su element za paragraf, naslov dokumenta, naslovi sekcija, liste… Kako se razvijao HTML, tako su se usporedo razvijali i preglednici koji su prikazivali HTML dokumente. Tvrtke su se natjecale kako bi što prije implementirale nove elemente i mogućnosti u HTML jezik da bi stekle prednost na tržištu. To je, naravno, uzrokovalo to da je HTML jezik bio različit ovisno o tome za koji se preglednik upotrebljavao, a bilo je i mnogo grešaka u vezi s novim elementima jer su programeri koji su radili za te tvrtke bili pod pritiskom da što prije razviju nove elemente za jezik. Programiranje web-stranica u to je vrijeme bilo kaotično i bile su potrebne stalne izmjene kako bi stranica radila, jer su konstantno dolazile nove verzije preglednika, a i promjene nad elementima u jeziku. Zbog tog je razloga Tim Berners-Lee osnovao World Wide Web Consortium (W3C) kako bi se kreirali standardi za

HTML i ostale web-jezike. Trebalo je neko vrijeme i nekoliko verzija jezika da bi se došlo do stabilne verzije, verzije 4. Nakon toga burnog razdoblja bilo je potrebno definirati semantiku i sintaksu jezika, što koji element označuje i kako se sintaktički točno piše. Da bi se opisala sintaksa HTML jezika, uporabljen je metajezik SGML (Standard Generalized Markup Language). SGML je dopuštao neke varijacije u sintaksi jezika, kao što je ispuštanje završne oznake (oznake zatvaranja), na primjer, nije bilo potrebno zatvoriti element za paragraf, što je uzrokovalo sporije učitavanje i prevađanje jezika, jer su preglednici morali prepoznati da nedostaje završna oznaka. Godine 1998. predstavljen je novi jezik kojim se mogla opisati sintaksa HTML jezika, a taj se jezik zove XML (EXtensible Markup Language) koji je sintaktički mnogo stroži od SGML-a i nije dopuštao izostavljanje završnih oznaka (svi elementi moraju imati završnu oznaku, a za one koji imaju samo jednu oznaku, ta oznaka mora biti završna) i još mnoge stvari koje programiranje u HTML jeziku čine mnogo konzistentnijim. Verzija HTML jezika čija je sintaksa definirana XML jezikom naziva se XHTML-om. Popularnost XHTML-a smanjila se nakon izlaska HTML5 verzije jezika [1].

2.1.2.Sintaksa i oblik dokumenta

Sintaksa HTML jezika vrlo je jednostavna, a sve je usredotočeno na oznake. Početna oznaka izgleda ovako `<p>`, a završna ovako `</p>` , ali nemaju svi elementi i početnu i završnu oznaku. Za otvaranje oznaka rabi se znak manje a za zatvaranje znak veće, a između se nalazi naziv elementa, a, ako je riječ o završnoj oznaci, i kosa crta. Svi elementi imaju takvu sintaksu i unutar njih (oni koji imaju početnu i završnu oznaku) u većini slučajeva može upisati neki sadržaj, a taj sadržaj mogu biti i drugi elementi, uz napomenu da se mora paziti jer ne mogu svi elementi sadržavati sve elemente. Elementi mogu imati atribute koji ih dodatno opisuju, a jedan od atributa koji svi elementi mogu imati jest ID koji jednoznačno identificira neki element. Svaki dokument započinje s `<!DOCTYPE html>`, što označuje da je riječ o HTML dokumentu, a nakon toga slijedi glavni element `<html>` unutar kojeg se nalaze svi ostali elementi. Prvi unutar tog elementa jest `<head>` unutar kojeg se može staviti naslov stranice i metapodatci kojima se može definirati autora, charset (što označuje skup znakova koji mogu biti šifrirani (eng. encode), odnosno znakovi koji se mogu koristiti), uključiti stilove ili druge datoteke (što može uključivati i JavaScript datoteke). I zadnje što ostaje jest `<body>` u kojem se nalazi sav sadržaj koji je prikazan na samoj stranici.[2]

2.1.3. HTML 5

HTML5 najnovija je verzija HTML jezika koja je sa sobom donijela velik broj promjena, nekih većih i nekih manjih. Cilj je verzije bio pojednostavniti neke stvari i dodati neke nove mogućnosti. Neke od manjih promjena koje su očite na svakom dokumentu kao što je promjena vezana uz `<!DOCTYPE>`. Promjena je iz `<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">` u `<!DOCTYPE html>`, što je jednostavnije za pamćenje i sam programer se ne mora brinuti o verziji ni o datoteci koja identificira taj standard za tu verziju. Neke druge mogućnosti koje spomenuta verzija nudi jesu: spremište na strani klijenta, što omogućuje da korisnik pristupi stranici kada nije spojen na internet (ali, naravno, taj je pristup limitiran), nema više potrebe za priključcima (eng. plug in), nego postoji podrška za mnoge oblike multimedija u samom HTML jeziku, mnoge nove mogućnosti s CSS-om itd. Naravno, s obzirom na to da je većina današnjih stranica dinamična, i sam se jezik sve više prilagođuje tom obliku stranica [3].

2.2. CSS

Na početku će ukratko biti opisana povijest samog jezika, a nakon toga sam jezik.

2.2.1. Povijest

Sredinom 90-ih godina prošloga stoljeća popularnost interneta je rasla pa se broj korisnika koji nisu bili vezani za obrazovne ustanove i istraživanja povećavao. S vremenom su korisnici počeli zahtijevati veću kontrolu nad prezentacijom HTML dokumenata jer sam HTML nije bio dovoljan za ono što su korisnici htjeli. Stilski su listovi oduvijek bili planirani uz sam HTML jezik, ali u to vrijeme još nisu bili standardizirani pa je uloga implementacije stilova pala na preglednike. Ubrzo nakon osnivanja W3C konzorcija Hakon Wium Lie objavljuje svoju prvu skicu kaskadnih listova. U njegovu radu na stilskim listovima pridružuje mu se Bert Bos koji je također imao svoj prijedlog za stilske listove. Zajedno su kreirali CSS koji na početku nisu prihvaćali svi, ali ga je internetska zajednica većinom prihvatila.[29]

2.2.2. CSS jezik

U HTML jeziku se može odrediti semantika dokumenta, njegov sadržaj i do neke razine njegova prezentacija, za više opcija oko samog izgleda i prezentacije dokumenta postoji CSS (kaskadni stilski listovi, eng. Cascading Style Sheets). Ti se stilovi mogu primijeniti na većinu elemenata

(oni elementi čiji sadržaj ili oni sami zauzimaju prostor na stranici). Elementima se mogu mijenjati boja, veličina, pozicija, vidljivost itd. Velika prednost primjene stilskih listova jest u tome što omogućuju odvajanje prezentacije od samog sadržaja i strukture dokumenta (moguće je mijenjati i strukturu tako da se mijenjaju pozicije elemenata). Svi se stilovi mogu pisati u zasebni dokument koji se zatim uključuje u HTML dokument preko `<link>` elementa u glavi dokumenta. Separacija prezentacije od sadržaja omogućuje da postoji više stilova po stranici te se samim time izgled može prilagoditi željama i/ili potrebama korisnika (stil koji se korisniku sviđa i/ili stil koji omogućuje lakše čitanje ljudima koji imaju problema s vidom). Također mogu postojati različiti stilovi za različite uređaje i veličine ekrana. Naravno, ništa ne sprečava to da se stilovi definiraju u samom HTML dokumentu, ali je bolje stilove odvojiti u zasebni dokument [2].

U HTML dokumentu stilovi mogu biti unutar `<style>` elementa ili unutar samog elementa kao njegov atribut. Naravno, pristup s atributom je loš jer, ako je potrebno mijenjati stilove, mora se naći element tako da se pretražuje cijeli dokument koji može biti dugačak. Ako se uporabljuje CSS u zasebnom dokumentu ili unutar `<style>` elementa, može se mijenjati stil svih elemenata, primjerice moguće je svim paragrafima promijeniti veličinu fonta ili postaviti pozadinu za cijeli `<body>` element. To dodjeljivanje stilova svim elementima nekog tipa omogućuje kreiranje predložaka koji se mogu rabiti za više HTML dokumenata, čime se olakšava dizajniranje stranice, na primjer moguće je imati jedan CSS dokument za sve stranice i po jedan za svaku stranicu. Uporaba zajedničkog CSS-a daje neku konzistentnost stranici, što mnogi korisnici cijene. Naravno, ako se želi promijeniti baš neki konkretni element ili grupa elemenata, moguće je primijeniti različite selektore. Postoji mnogo selektora, ali se najčešće se primjenjuju selektori s pomoću ID-a, klase ili roditelja. CSS omogućuje mijenjanje izgleda nekog elementa kao odgovor na neki događaj kao što je klik ili sama prisutnost miša nad elementom, na primjer moguće je povećati link kad je miš nad njim [2].

Što se tiče same sintakse, ona je jednostavna. Prvi je selektor nakon kojeg se otvara vitičasta zagrada i unutar otvorenoga bloka zapisuju se svojstva koja se sastoje od imena svojstva i njegove vrijednosti i na kraju je zatvorena vitičasta zagrada. Naravno, na početku može biti i više selektora ako se želi da se stil odnosi na više elemenata [2].

2.3. JavaScript

JavaScript je programski jezik koji služi za programiranje na strani klijenta. S pomoću njega može se izvršiti provjera podataka prije nego se pošalju podatci na server te, ako ti podatci nisu u redu, ne mora se slati zahtjev na server (može se zaustaviti događaj koji bi inače slao zahtjev ili AJAX zahtjev ako se to primjenjuje), čime je moguće znatno reducirati broj poziva na server i samim time ubrzati aktualna stranica. Naravno, sam jezik omogućuje mnogo više od toga. Moguće je pomoću njega mijenjati stil elemenata u dokumentu i tako primjerice napraviti interaktivni obrazac koji govori korisniku je li dobro ispunio obrazac te, ako nije, gdje su greške. Osim mijenjanja stilova elemenata, mogu se mijenjati i njihovi atributi, primjerice moguće je staviti da je unos za ponovljenu lozinku samo za čitanje sve dok korisnik ne unese prvu lozinku, a zatim se dopušta unošenje ponovljene lozinke. Još jedna zanimljiva stvar koju jezik nudi jesu skočni prozori kojima je moguće obavještavati, upozoravati i prikazivati greške korisnicima. Jezik također može „hvatati“ događaje te na temelju njih izvršavati provjere ili neke druge radnje koje programer želi da se izvrše dok korisnik nešto napravi, a neki od tih događaja jesu: na klik mišem, na prijelaz miša, na izlaz iz polja za unos, na unos novog znaka u polje za unos itd. JavaScript može mijenjati sadržaj dokumenta u smislu da može kreirati nove elemente (cjeline, paragrafe, obrasce, gumbove, …). To zajedno s komunikacijom i dohvaćanjem podataka sa servera omogućuje izradu dinamičnih stranica čiji se sadržaj mijenja ovisno o tome što korisnik radi [1].

2.3.1. Povijest

Brendan Eich je kreator JavaScripta koji se u ranijim danima razvoja nazivao LiveScript. Na kraju 1995. godine prije službenog puštanja jezika u javnost jezik je bio preimenovan u JavaScript, čime su pokušali kreirati poveznicu sa sve popularnijim jezikom Java, ali sam JavaScript nije sličan Javi. Sam JavaScript bio je pušten uz preglednik Netscape 2.0, ali je postao popularniji tek svojom verzijom 1.1 koja je bila puštena uz Netscape 3.0. Ta je verzija već imala reagiranje na događaje kao što je na primjer prelazak preko nekog elementa. Bilo je još verzija koje su izlazile uz sam preglednik. Microsoft je imao vlastiti jezik koji je imao istu ulogu kao i JavaScript, a naziva se JScript i razvijao se zajedno s Microsoftovim preglednikom Internet Explorer. Usto je postojala još jedna inačica jezika koja se nazivala ECMAscript i koju je na zahtjev tvrtki koje posjeduju JavaScript napravila europska asocijacija proizvođača računala (ECMA, eng. European Computer Manufacturers Association) sa svrhom da se standardizira JavaScript. Ovaj je pokušaj imao uspjeha u zbližavanju dvaju jezika, JavaScripta i JScripta.

Danas većina preglednika podržava ECMAscript standard i JavaScript jezik može se rabiti na njima [1].

2.3.2. JavaScript jezik i njegova sintaksa

JavaScript jezik je interpreterski jezik, što znači da nema potrebe za kompiliranjem, već postoji interpreter koji čita i izvršava takav program. To je, s jedne strane, prednost jer, kada se modificira sam kod, nije potrebno kompilirati, već samo spremiti dokument, a i uobičajeno ima manje datoteka jer ne postoji kompilirana verzija dokumenta. Interpreterski su jezici također uobičajeno jednostavniji od kompilatorskih i samim time zahtijevaju manje linija da bi se nešto napisalo. Nedostatci su takvog jezika u tome što kada se pokrene datoteka s kodom, interpreter ju mora prevesti i izvršiti, što uobičajeno traje dulje nego pokretanje kompiliranog programa jer je on već kompiliran i spreman za izvršavanje. Naravno, sama jednostavnost koda isto tako nije bez cijene, JavaScript kod je u usporedbi s kompilatorskim jezicima dosta manje efikasan, primjerice JavaScript ima samo jedan generički numerički tip podataka i zbog toga je izvršavanje aritmetičkih operacija manje optimalno. Usto, budući da nema kompiliranja, ne može se znati postoje li greške sve dok se program ne pokrene, ali i onda neke greške nisu prepoznate. JavaScript je napravljen tako da bude što fleksibilniji i, imajući to na umu, u JavaScriptu nema potrebe za deklariranjem varijabli, što znači da se lagano može kreirati nova varijabla koja nije trebala biti kreirana, tako da se pogrešno napiše ime varijable koja je već bila kreirana i takva greška neće biti prepoznata (i usto, sam jezik osjetljiv je na mala i velika slova). Same greške najčešće se mogu vidjeti u samim preglednicima, uz napomenu da mora biti na raspolaganju alat koji omogućuje prepoznavanje grešaka. JavaScript je također skriptni jezik, što znači da služi za automatiziranje zadataka u nekom softverskom okruženju (u ovom slučaju to su preglednici). Inače je riječ o jednom softverskom okruženju, ali u slučaju jezika JavaScript ima ih mnogo i zbog toga je svaka implementacija spomenutog jezika podijeljena na dvije komponente. Prva se sastoji od intepretera i glavne funkcionalnosti ECMAscripta i ona je neovisna o tome o kojem se okruženju riječ, odnosno mora biti prisutna kod svih. Druga komponenta sadržava sve dodatne mogućnosti koje omogućuje samo okruženje. Sve mogućnosti koje nudi specifično okruženje u obliku su objekata .[1]

Sama sintaksa jezika donekle je slična C++ jeziku, na kraju svake linije koda mora biti „;“, petlje se pišu unutar vitičastih zagrada, na isti se način kreiraju funkcije itd. Budući da se varijabla ne treba deklarirati, ne može joj se dodijeliti tip, odnosno njezin se tip dodjeljuje automatski s obzirom na vrijednost koja joj je dodijeljena, naravno, vrijednost se dodjeljuje s pomoću znaka

jednako. Operatori za aritmetičke operacije su standardni, a komentari su dvije kose crte ili jedna kosa crta i zvjezdica za više redova. [4]

Kako bi se koristio JavaScript i njime mijenjao HTML dokument, mora ga se nekako uključiti u sam dokument. Sam kod može biti unutar samog dokumenta u glavi ili u tijelu dokumenta, a može biti i u drugom dokumentu. Neovisno o načinu koji je odabran za uključivanje, rabi se element `<script>` unutar kojeg se može pisati kod ili staviti izvor (dokument u kojem se nalazi kod). [4]

2.4. Razvojne okoline

Razvojne okoline općenito služe kako bi se ubrzao proces razvoja u nekoj tehnologiji jer većina se jezika može napisati u bilo kojem uređivaču teksta; naravno, da bi se izvršio sam kod, uglavnom je potrebno neko okruženje u kojem će se program izvršiti. Kod web-tehnologija to je preglednik, što znači da neko razvojno okruženje zapravo nije obvezno jer se sve može napisati u uređivaču teksta i onda pokrenuti na pregledniku. Osim toga što nude izvršavanje koda, razvojne okoline često služe i za ujedinjavanje više tehnologija koje uobičajeno služe za razvoj neke vrste aplikacija, na primjer web-aplikacije (PHP,HTML,CSS i JavaScript). Jedan od načina da se ubrza proces razvoja jest da se pojednostavni samo programiranje, a najčešći načini za to jesu predlaganje završetka neke linije i dijela linije koda (npr. kada se upiše neki objekt, da ponudi sve njegove funkcije i varijable tako da se ne mora ručno pisati ili, ako postoji neka varijabla, da je kasnije u tekstu prepozna i ponudi ako se uporabljuje negdje drugdje), generiranje blokova koda kao što su petlje, uvjeti ili čak i cijele klase na osnovi nekog modela ili nekog drugog izvora. Većina razvojnih okolina također ima mogućnost prepoznavanja grešaka i predlaganja mogućih rješenja za te greške, a u najmanju ruku pruža sažet opis greške i, naravno, njezinu lokaciju. Usto, mnoge okoline imaju i neki način da se program izvršava liniju po liniju i tako programer može pratiti rad programa i preciznije i brže naći gdje se nalazi neka logička greška. Još jedna korisna stvar koja se često pojavljuje jest kodiranje gdje se može vidjeti više datoteka odjednom, tj. programirati tako da se može gledati neki drugi kod koji je nekako povezan s kodom koji se trenutačno piše (npr. može se gledati neka klasa koja je trenutačno u uporabi koja se nalazi negdje drugdje ili je moguće gledati HTML dokument kada se piše JavaScript kod kako bi se lakše vidjelo što i gdje se mijenja). Većina okolina također ima negdje prikazan skup svih datoteka i mapa koje se nalaze u projektu, što omogućuje lako strukturiranje projekta i laku navigaciju među datotekama (pod pretpostavkom da je projekt dobro strukturiran, što većinom znači da su u mape grupirane datoteke koje služe istoj svrsi, na

primjer sve CSS datoteke u jednu mapu unutar koje postoji više mapa koje se odnose na pojedine stranice). Većina razvojnih okruženja ima više verzija, u smislu da imaju verzije koje su besplatne i koje se plaćaju. Nasreću, besplatne su verzije uglavnom i više nego dovoljne za većinu onoga što se radi tijekom programiranja (većina prije nabrojenih mogućnosti pripada besplatnim verzijama). Plaćene verzije većinom sadržavaju mnoge napredne mogućnosti koje mogu znatno skratiti neke aspekte programiranja, ali su pokatkad i poprilično zahtjevne za uporabu i zahtijevaju dublje znanje nekog jezika ili općenito programiranja.
Kod web-tehnologija takve okoline znatno pomažu u tome da se ne moraju pamtiti nazivi svih atributa i naredbi, pri pronalaženju grešaka, što je vrlo korisno jer je lako moguće izgubiti se u svim silnim oznakama HTML jezika, mnogim rukovateljima događaja u JavaScript jeziku. Nažalost, većina okolina još uvijek ne može savršeno otkriti greške unutar JavaScript koda jer ima mnogo osobina koje to onemogućuju (varijable nemaju fiksni tip, ne moraju se deklarirati itd.) pa je zbog toga vrlo korisna mogućnost prelaženja koda liniju po liniju.
U nastavku će biti opisana neka besplatna razvojna okruženja koja se mogu iskoristiti za izradu web-aplikacija.

2.4.1. NetBeans

To je razvojno okruženje kojemu je primarna svrha razvoj Java-aplikacija te stolne i web-aplikacije. Podržava najnoviju inačicu Java jezika. To okruženje također podržava HTML5, CSS, JavaScript, PHP,C i C++ te je otvoreni kod (eng. open source), što znači da to okruženje nadograđuje zajednica koja se njime koristi. To okruženje nudi automatsko poravnanje koda, uvlačenje koda kako bi se vidjela njegova hijerarhija, semantički i sintaktički smisleno obojen kod, mogućnost lakog refaktoriranja koda, savjeti za programiranje, generiranje koda i predlošci koje korisnik može koristiti kako bi si skratio posao. Također nudi verzioniranje koda preko popularnih alata za verzioniranje kao što je Git. Usto, postoji i mogućnost kreiranja sučelja koristeći povuci i pusti način rada. Podržava i debagiranje koda liniju po liniju i usto ovo okruženje podržano je na svim najpoznatijim operacijskim sustavima, a zbog toga što je otvoreni kod, postoje mnogi priključci koje je stvorila zajednica i koje je moguće koristiti.[5]

2.4.2. RJ TextEd

RJ TextED je uređivač teksta koji ima veliku podršku za razvoj web-aplikacija. Podržava najčešće jezike koji se rabe za razvoj web-aplikacija (PHP, ASP, JavaScript, HTML i CSS). Ima mnogo mogućnosti, pa će biti nabrojene samo one koje ovo okruženje čine drukčijim. Ima mogućnost pregleda kako izgleda HTML dokument zajedno sa svojim CSS-om, ima ugrađenu

HTML5 validaciju, mogućnost uspoređivanja kodova, mogućnost istodobnog pregleda više datoteka i mnoge predloške. Također ima vlastiti (S)FTP (file transfer protocol, protokol za prijenos datoteka) klijent koji nam omogućuje učitavanje datoteka. [6]

2.4.3. Brackets

Brackets je uređivač teksta kojemu su glavna obilježja to da je lagan, ali i sadržava sve bitne alate za razvoj web-aplikacija. Mogućnosti koje su najviše naglašene jesu: pisanje HTML koda uz mogućnost paralelnog pisanja i pretraživanja CSS koda (pretraživanje po ID-u elementa primjenom kratice), automatsko ažuriranje preglednika i dobro podržan rad s pretprocesorima.[7]

3. ASP.NET

Prije samog opisa ASP.NET-a prvo će biti opisan jezik u kojem je izrađen, a to je C#.

3.1. C# programski jezik

.NET okvir i jezik C# predstavila je tvrtka Microsoft 2002. godine i zajednica ih je brzo prihvatila. .NET okvir primarno je namijenjen za rad na Windows operacijskim sustavima, ali je moguć i na nekim drugim sustavima poput Linuxa. Neka dobra obilježja tog okvira jesu: interoperabilnost koda, podržavanje raznih jezika (poput Visual Basic jezika, F#, itd.), integracija između jezika (klasa u C# se može koristiti u nekom drugom jeziku), velika baza klasa (postoje klase za razne probleme i razne tipove podataka) itd. [8].

C# jezik je objektno orijentirani kompajlerski jezik koji je moćan i fleksibilan, a po sintaksi je sličan Java programskom jeziku. Razlog tome je što su oba jezika bazirana na C-jezicima. C# ima sličnosti s Visual Basic i C++ jezicima. C# također uporabljuje LINQ (eng. Language integrated query, upiti integrirani u kod) koji, kao što i samo ime govori, omogućuje uporabu upita u kodu. C# je mješavina raznih jezika te je zbog toga relativno jednostavan, ima čistu sintaksu i vrlo je moćan i fleksibilan. Sada će biti nabrojane neka dobra obilježja tog jezika [8].

C# ne treba pokazivače, oni nisu potrebni, ali se mogu upotrebljavati ako je zbog nekog razloga to neizbježno. Automatsko upravljanje memorijom eliminira potrebu za naredbom `delete` jer postoji automatsko sakupljanje otpada. Tu su formalni sintaktički konstrukti za klase, sučelja itd. Podržava atributno programiranje, odnosno može se dodatno opisati ponašanje tipova i njihovih članova tako da je lakše znati čemu oni služe, jesu li ažurni itd. Postoji mogućnost kreiranja generičkih tipova. Podržava anonimne metode i tipove. Definiranje tipa čija domena seže između različitih datoteka koda. Tu je i već spomenuti LINQ koji omogućuje različite manipulacije podatcima. Povećava funkcionalnost postojećih tipova. Postoji lambda-operator. Omogućuje popunjavanje svojstva objekta pri kreiranju tog objekta. Relativno je jednostavno programiranje s više dretava. Naravno, postoji još mnogo bitnih obilježja, ali su ovo sva koja će biti ovdje nabrojana. C# jezik je napravljen tako da može raditi samo unutar .NET okvira, ali to ne bi smjelo biti problem jer okvir podržava razne operacijske sustave [8].

C# za razvoj web-aplikacija primarno je jezik na strani servera te se za to može upotrebljavati. To je solidan jezik, ali na kraju krajeva to je samo jezik i sve što se može napraviti u C# jeziku može se napraviti i u drugim jezicima na strani servera, samo je pitanje načina kako doći do istoga rješenja. C# je objektno orijentiran i postoje mnoge klase koje smanjuju količinu koda

koja je potrebna da bi se nešto napravilo, ali naravno postoje i komplikacije i ograničenja u vezi s tim obilježjem. Uporaba C# jezika ispred drugih jezika ovisi o potrebama i znanju samog programera, odnosno, ako programer ima znanje u C# jeziku, definitivno će mu biti lakše raditi u njemu, ali, ako programer nema znanja ni o jednom jeziku za programiranje na strani servera, tada se izbor svodi na osobne preferencije. Moglo bi se reći da je C# odličan zbog toga što u kombinaciji s razvojnim okruženjem Visual Studio nudi razne mogućnosti i svakakve prečace u kreiranju koda, ali to nije izravno vezano za sam jezik koji nije vezan za to razvojno okruženje (naravno, dobro okruženje koje podržava jezik može biti razlog za odabir jezika). Sljedeća stvar jest činjenica da je ASP.NET većinski napisan u C# jeziku i potrebno je znanje C# jezika kako bi se lakše upotrebljavao i, naravno, sam jezik na strani servera tada mora biti C#. ASP.NET okvir nudi mnogo dobrih mogućnosti, a jedna je od njih i MVC uzorak koji automatski kreira dosta toga što je potrebno za web-aplikaciju i ima strukturu kojom, ako je korisnik prati, može relativno jednostavno napraviti web-aplikaciju. Na kraju, C# za web razvoj donosi okvire koji su napravljeni u njemu, ali sam jezik dobar je za uporabu na strani servera, kao i ostali popularni jezici za razvoj na strani servera koji, sigurno, također imaju razne okvire koji olakšavaju izradu web-aplikacija. Može se zaključiti da će programeru, ako je već upoznat s C# jezikom ili njegovom razvojnom okolinom, biti lakše naučiti ASP.NET i raditi web-aplikacije u tom okviru [24].

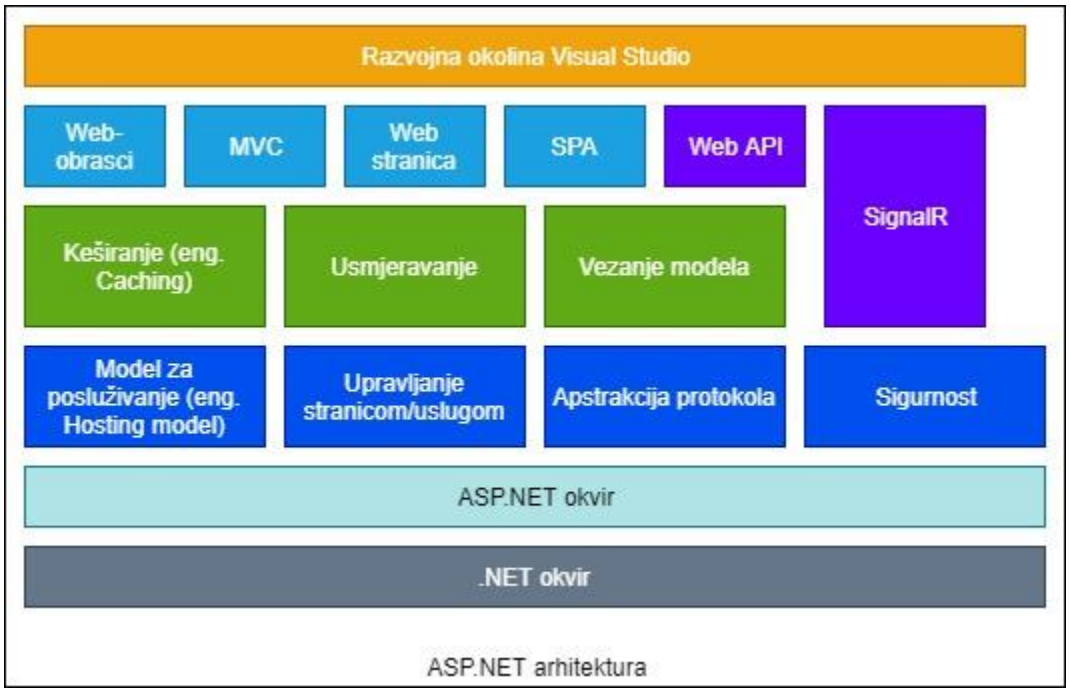
3.2. Osnovni koncepti

APS.NET je relativno novi dodatak .NET okviru čija je prva verzija bila puštena 2007. godine. Taj je dodatak tražila zajednica, poglavito ALT.NET pokret koji je htio okvir koji bolje i lakše radi sa samim HTTP protokolom, koji je lakše testirati i pri kojemu postoje jasno odijeljeni dijelovi tako da je lakše snaći se u strukturi projekta i lakše je programirati (eng. Separation of concerns) [8].

Prije MVC-a programeri koji su htjeli raditi web-aplikacije koristeći se .NET okvirom morali su upotrebljavati ASP.NET web-obrasce (eng. web-forms) koji su izašli više od pola desetljeća prije ASP.NET MVC okvira. Postavlja se pitanje zašto bi Microsoft razvijao novi okvir umjesto da unaprijedi ili promijeni stari. U vrijeme nastanka web-obrazaca web razvoj još nije bio savršeno utvrđen, a i sama skupina ljudi koja se koristila .NET tehnologijom nije bila naviknuta na izradu web-aplikacija, pa je Microsoft odlučio kreirati web-obrasce tako da sadržavaju mnogo koncepta stolnih aplikacija i to je bilo vrlo uspješno. Mnogo je programera prešlo na web i nastala je zajednica koja je pridonosila samom okviru. S vremenom, kako su se razvijale web-tehnologije,

sve je više programera htjelo imati veću kontrolu, što im web-obrasci nisu mogli omogućiti jer bi to zahtijevalo velike promjene u samoj jezgri tog okvira. Microsoft je odlučio napraviti ASP.NET MVC kako bi udovoljio zahtjevima spomenutih programera i zadržao već postojeću zajednicu koja je naviknula na web-obrasce. ASP.NET MVC ima mnoga obilježja koje kodiranje u okviru čine čišćim, ali istodobno zahtjeva više znanja o HTML i JavaScript jezicima i HTTP protokolu. Također jedna od polica ASP.NET MVC jest konvencija iznad konfiguracije, a to znači da postoji više konvencija (npr. specifična struktura mapa, odnosno projekta) koje kao posljedicu imaju manju potrebu za konfiguracijom same aplikacije [8].

Prije samog opisa MVC uzorka, prikazana će biti arhitektura ASP.NET okvira.



Slika 1. Arhitektura ASP.NET okvira (napravljeno prema: https://www.dotnettricks.com/learn/aspnet/understanding-detailed-architecture-of-aspnet-45)

Sam opis arhitekture početi će s .NET okvirom. Okvir je ugrađen u Windows operacijske sustave i omogućuje razvoj i izvršavanje aplikacija koje su napravljene u Microsoftovim okvirima i tehnologijama (tu naravno spada i ASP.NET okvir). [30]

ASP.NET okvir se koristi za razvoj web aplikacija te je nadograđen na sam .NET okvir. Okvir pruža razne mogućnosti koje su na slici prikazane tamno plavom bojom (model za posluživanje, sigurnost…). U ovom će opisu okvir biti podijeljen na dva glavna dijela, ASP.NET usluge (eng. services) i ASP.NET stranice. [30]

Postoje dvije vrste ASP.NET usluga od kojih je prva web API. Radi se o okviru za izradu HTTP usluga (te usluge se mogu koristiti na preglednicima, mobitelima, itd.). Druga vrsta je SignalR biblioteka koja pojednostavljuje dodavanje mogućnosti weba u stvarnom vremenu (eng. real-

time web). Radi se o mogućnosti koja kada dođe do promjene u sadržaju (na serveru), automatski šalje sadržaj svim spojenim korisnicima (nije potreban zahtjev korisnika). [30]

Što se tiče ASP.NET stranica, postoje četiri modela razvoja. Prvi model su web-obrasci, radi se o razvoju temeljenom na događajima. Razvoj u ovom modelu je vrlo sličan razvoju desktop aplikacija u Visual Studio okruženju i ovaj model je već bio ranije spomenut u radu. Drugi model je MVC koji će se koristiti za izradu aplikacije u radu i biti će detaljno opisan kasnije. Ovaj model omogućuje brzi i agilni razvoj web aplikacija. Treći model su web stranice, radi se o laganom modelu koji je baziran na Razor sintaksi (sam Razor stroj bit će opisan kasnije). Koristi se za razvoj web aplikacija prema najnovijim web standardima te u sebi ima ugrađene razne predloške i pomagače koji olakšavaju sam razvoj web aplikacije. Zadnji model je SPA (eng. Single Page Application, aplikacija na jednoj stranici). Radi se o tehnologiji gdje se stalno dinamički mijenja sadržaj jedne stranice umjesto da se učitavaju nove stranice sa servera. Ovaj model primarno služi za izradu web aplikacija koje uključuju čestu komunikaciju na strani klijenta. [30]

I zadnje što će biti spomenuto je Visual Studio okruženje koje se može koristiti za razvoj web aplikaciju u svim gore navedenim modelima.

3.2.1. MVC uzorak

Sam uzorak postoji već neko vrijeme, ali je tek nedavno zaokupio veću pozornost i pojavljuje se u mnogim okvirima i programskim jezicima. Neki su od njih: Java (Spring okvir), Ruby, neki JavaScript klijentski okviri (Angular, EmberJS) i, naravno, .NET okvir. Sam naziv stoji za model-pogled-kontroler (engl. Model-View-Controller), a to su ujedno i glavna tri dijela na koje je podijeljen ovaj uzorak [8].

Modeli su zapravo podatci aplikacije, koji su većinom obični CLR (engl. Common Language Runtime) objekti. Model za pogled može se sastojati od više modela koji su namijenjeni za neki specifičan pogled. Modeli se mogu predočiti kao tablice u bazi podataka, a modeli za pogled kao pogledi u bazi podataka (jedan pogled može dohvaćati podatke iz više tablica i često se kreiraju za upite koje korisnici često postavljaju kako bi se, umjesto da se svaki put piše isti upit, jednostavno mogao iskoristiti model za pogled). U teoriji bi modeli trebali biti čisti, što znači da ne bi smjeli sadržavati nikakva poslovna pravila ni provjere, nego samo čiste podatke, ali, naravno, u praksi to ovisi o mnogim stvarima kao što je sam programski jezik koji se upotrebljava ili sam okvir koji se uporabljuje [8].

Pogledi su ono što je prikazano korisniku, oni čine korisničko sučelje. Budući da su sučelje, oni bi trebali samo primati podatke i naredbe prema kojima zatim prikazuju korisniku podatke i ne bi smjeli sami obrađivati podatke, nego bi za to bili zaduženi kontroleri [8].
Kontroleri su glavni i omogućuju da sve zajedno funkcionira. Oni primaju zahtjeve od korisnika i obrađuju ih i po potrebi šalju drugim komponentama te se brinu o tome da bi se, ako je potrebno, promjene uzrokovane samim zahtjevima korisnika prikazale na pogledima [8].

3.2.2. Struktura projekta

Sama struktura projekta se sastoji od nekolicine mapa od kojih svaka čini neki dio (komponentu) projekta, svaka je od njih smislena cjelina i tako je ostvarena razdvojenost koncepata koja je jedna od praksi čistog koda. Važno je napomenuti da sve datoteke imaju svoje mjesto i da se to ne smije mijenjati da bi aplikacija dobro funkcionirala i to je jedna od konvencija koje se treba držati. Nisu sve datoteke unutar mapa, nego postoje i one koje su izvan, odnosno u samom korijenu projekta, pa će one biti nabrojene u nastavku. Favicon.ico je ikona koja se pojavljuje pokraj imena stranice i bitno je da ona postoji ili će se smanjiti performansa stranice jer je preglednik očekuje. Global.asax.cs čini ulaznu točku u aplikaciji. Ako se ništa ne mijenja, Global.asax.cs klasa koristi se samo jednim rukovateljem događaja i taj je događaj samo pokretanje aplikacije. Uz njega postoje mnogi drugi od kojih će biti nabrojeni oni koji se najčešće uporabljuju, a to su zatvaranje aplikacije, pojava greške (omogućuje da se reagira na greške i javi korisniku što se dogodilo), početak i kraj sesije i početak i kraj zahtjeva. Svi su ti događaji na razini aplikacije, što može biti vrlo korisno jer je moguće kreirati generalne odgovore na neke događaje na razini aplikacije. Packages.info sadržava konfiguracijske podatke vezane u NuGet pakete koji se uporabljuju u samoj aplikaciji (NuGet paketi mogu biti vrlo korisni te olakšati rad na aplikaciji ili dodati neke nove mogućnosti i kontrole). AplicationInsights.config služi za konfiguriranje uvida. Startup.cs je klasa koja služi za otvoreno (eng. open) web-sučelje, a njome se koristi ASP.NET identitet (eng. identity). Web.config je konfiguracijska datoteka za projekt [8].
Sada će biti objašnjene neke mape koje se automatski kreiraju s projektom počevši od mape za modele koja služi za sve klase koje čine modele. Preporučuje se da ta klasa bude u zasebnom DLL-u (engl. Dynamic Link Library, biblioteka s dinamičnim povezivanjem) ako je riječ o većem projektu koji ih može imati na stotine [8].
Mapa za kontrolere sadržava sve kontrolere. Mapa za poglede sadržava sve poglede. Još jedna od konvencija MVC uzorka jest da za svaki kontroler postoji određeni pogled koji ima isto ime kao i kontroler, naravno, bez dijela kontroler (npr. kontroler ForumController trebao bi imati pogled Forum u datoteci za poglede, u zasebnoj mapi istog imena, što znači da svaki kontroler

ima svoju mapu i da jedan kontroler može imati više pogleda). U samom korijenu pogled mape nalazi se web.config koji definira osnovni tip stranice i u projektima koji rabe Razor stroj (eng. engine) služi za dodavanje referenca i izjava o uporabi (engl. using statemant, linije koda koje govore da se u projektu uporabljuje neki tip iz određenog prostora imena), koji će biti detaljno objašnjen kasnije. Također se nalazi i _ViewStart.cshtml koji se pokreće prije svih pogleda i specificira koja je shema (engl. Layout, isto je pogled) pogleda zadana, ako nijedan drugi nije specificiran (taj se pogled može rabiti za elemente koje sadržavaju svi pogledi, npr. logo aplikacije) [8].

Dijeljena mapa (Shared folder) nalazi se unutar mape za poglede i toj mapi mogu pristupiti svi pogledi. U njoj se nalaze zadana shema (engl. Layout, _Layout.cshtml) i zadani predložak u slučaju greške (Error.cshtml) [8].

Također postoje i mape koje su za sami ASP.NET. Neke su od njih automatski uključene u projekt, a neke nisu. One koje nisu uključene moguće je kreirati kao i normalne mape u projektu (desni klik i dodaj (engl. add)), samo što je potrebno odabrati ASP.NET web-folder te točno željenu vrstu mape (svaka vrsta ima svoju ulogu) i potvrditi stvaranje. Ovakve mape služe za spremanje tema, koda, podataka, resursa itd. Sada će te mape biti nabrojene i kratko opisane.

Prva je App_Data u koju se stavljaju svi podatci koji su u obliku datoteka i koji su potrebni za rad same aplikacije [8].

Druga je App_Code koja sadržava datoteke s kodom te se taj kod dinamički kompilira (što znači da, ako se naprave promjene nad klasama u toj mapi, one će odmah biti vidljive u smislu da ih nije potrebno ponovno kompilirati, nego se to obavlja automatski, na primjer, ako se promijeni klasa koja uzrokuje promjenu koja je vidljiva, moguće ju je samo spremiti i osvježiti preglednik i ta bi promjene trebala biti vidljiva) [8].

Sljedeća je App_Themes u koju se stavljaju teme koje se uporabljuju u aplikaciji. Sljedeća je App_browsers u koju se smještaju datoteke vezane za rad u različitim preglednicima, odnosno datoteke kojima se sam ASP.NET koristi kako bi prepoznao o kojem je pregledniku riječ i koje su njegove mogućnosti [8].

Sljedeća je App_GlobalResources koja služi za spremanje datoteka s resursima koji su dostupni na razini aplikacije, a glavna uporaba te mape jest lokalizacija i globalizacija (omogućuje kreiranje aplikacije koja može biti na različitim jezicima kako bi bila razumljivija korisnicima iz raznih zemalja). I zadnja Mapa App_LocalResources također služi za spremanje datoteka s resursima, ali na razini neke stranice [8].

Uz sve te mape postoji još jedna mapa koja se naziva App_Start. Naime, kako je prije bilo spomenuto sav konfiguracijski kod stranica (osiguranje, usmjerivanje,…) nalazio se u jednoj

klasi koja se zove Global.asax.cs. Kako je s novim verzijama bilo sve više konfiguracija, postalo je nepraktično sve imati u jednoj klasi, pa je ta klasa bila podijeljena na više klasa, od kojih je svaka imala neki dio odgovornosti Global.asax.cs klase koji je činio smislenu cjelinu. Sve te klase nalaze se u App_Start mapi te će biti nabrojene i ukratko objašnjene, a poslije i malo detaljnije. Prva je klasa BoundleConfig.cs koja služi za kreiranje paketa za JavaScript i CSS datoteke, što je vrlo korisno, a zbog čega bit će objašnjeno poslije. Sam programer može kreirati pakete kako želi unutar te klase. FilterConfig.cs služi za registriranje filtra (posebna vrsta klase čija će uloga će biti poslije opisana) na razini cijele aplikacije. IdentityConfig.cs sadržava klase koje podupiru ASP.NET identitet (eng. identity). RouteConfig.cs je klasa u kojoj se konfigurira tablica usmjerivanja. Klasa Startup.Auth.cs služi kao ulazna točka za konfiguraciju ASP.NET identiteta [8].

Prva klasa BoundleConfig.cs služi za kreiranje paketa, ali ujedno i smanjivanje samih datoteka koje su dio paketa. Stvaranje paketa zapravo je spajanje više datoteka u jednu te se primarno primjenjuje kako bi se ubrzao rad aplikacije. Svaki preglednik ima ograničenje na broj datoteka koje mogu biti odjednom preuzete s nekog servera pa je zato bolje spojiti više datoteka u jednu kako ne bi bio prijeđen taj limit i usporio se rad aplikacije. Naravno, stavljanje svih datoteka u jedan paket isto tako nije pametno jer će se dugo preuzimati, trebalo bi podijeliti datoteke po stranicama i tako imati maksimalno nekolicinu paketa za svaku stranicu, čime bi se osiguralo da se ne prijeđe limit preuzimanja, a i da se paketi ne preuzimaju previše dugo zbog njihove veličine. Smanjivanje se također primjenjuje za ubrzavanje stranice, tj. učitavanja. To se postiže smanjivanjem koda JavaScript i CSS datoteka. Naime, sve su varijable uobičajeno po pravilima čistog koda imenovane tako da se na temelju imena može zaključiti čemu služi ta varijabla (ta imena mogu biti dugačka kao na primjer `PrviProstiBroj`). Klase, ID-ovi itd. također imaju imena koja imaju smisla i na temelju kojih se može barem približno zaključiti o kojem je elementu ili skupini elemenata riječ (na koje se elemente odnosi neki stil). To je jedna od stvari koju smanjivanje mijenja, smanjivanje smanjuje imena varijabli na manji broj znakova (pokatkad i samo jedan znak) te uz ostale promjene koje radi nastoji smanjiti datoteku kako bi bila što manja i samim time se brže prenijela preko mreže. Uobičajeno uvijek postoje dvije verzije datoteka, jedna koja nije smanjena i jedna koja jest. Datoteka koja nije smanjena uporabljuje se tijekom razvoja jer je mnogo čitljivija, a smanjena se rabi kada se stranica pusti u javnost. Tomu služi klasa BundleConfig.cs. Ona upravlja postavkama na temelju kojih se kreiraju sami paketi i smanjuju datoteke, te su tu nabrojeni postojeći i mogu se kreirati novi paketi (s `bundles.Add`). Na početku, kad je projekt tek kreiran, zadano je da se odmah sve minimizira i stavlja u pakete u formatu za produkciju, ono što u to vrijeme postoji i, naravno, sve što programer poslije doda.

Ako se želi da se smanjuje i stavlja u pakete u modu za otklanjanje grešaka, moguće je to napraviti u samoj klasi ili u konfiguracijskoj datoteci Web.config.[8]

Sljedeća je klasa FilterConfig.cs koja služi za registriranje filtara na razini aplikacije. Filteri su klase koje omogućuju da se nešto napravi kao reakcija na neki zahtjev ili akciju, većinom vezano za neki trenutak, prije, tijekom i nakon izvršenja samog zahtjeva ili akcije, što znači da se te klase pozivaju kada se poziva određena akcija ili zahtjev. Te klase mogu služiti za neku određenu akciju (metodu) ili za cijelu kontroler klasu ili čak i za sve klase, odnosno globalno. To znači da je za neki tip zahtjeva koji zahtijeva da se nešto napravi prije nego što počne obrađivanje (npr. autorizacija) to je moguće globalno napraviti za taj tip zahtjeva s pomoću filtarske klase. Postoje četiri vrste filtarskih klasa koje će biti opisane u nastavku [8].

Prva vrsta implementira `IAuthorizationFilter` i logično, budući da je bitno znati ima li korisnik pravo slati neki zahtjev, taj se filtar izvršava prije svih ostalih. On, kao što i sam naziv kaže, služi za autorizaciju, a sustav funkcionira tako da neka klasa ima atribut koji pokazuje koju razinu zahtijeva (ako korisnik mora biti registriran i prijavljen, tada se rabi atribut `[Authorize]`, a, ako zahtjev može poslati bilo koji korisnik, tada se uporabljuje atribut `[AllowAnonymous]`) [8].

Druga implementira `IActionFilter` i služi samo za akcije i omogućuje da se nešto napravi prije akcije s `OnActionExecuting` i nakon što završi izvođenje same akcije s `OnActionExecuted`.[8]

Treća implementira `IResultFilter` i služi kako bi se moglo reagirati na rezultat neke akcije. Moguće reagirati prije nego što se vrati rezultat neke akcije s `OnResultExecuting` te nakon što je taj rezultat izvršen s `OnResultExecuted` [8].

Zadnja implementira `IExceptionFilter` i služi za rukovanje iznimkama. Sam ASP.NET ima zadani filtar koji pokreće stranicu koja pokazuje koja se greška dogodila, ali i sam programer može napraviti filtar za neku iznimku ili za sve njih (na globalnoj razini) [8].

Sljedeća je klasa IdentityConfig.cs koja zajedno s klasom Startup.Auth.cs služi kao podrška za ASP.NET identitet [8].

Slijedi RouteConfig.cs klasa koja služi za kreiranje različitih oblika URL-ova kako bi pojednostavnio rad korisnicima (prilagodio oblik da bude lakše razumljiv i pamtljiv). Ta klasa omogućuje kreiranje URL uzoraka s pomoću kojih ASP.NET MVC zna kojeg kontrolera i koju akciju mora izvršiti. Sam način kako se to radi u samom kodu bit će objašnjen pri kreiranju primjera web-stranice [8].

U font mapi nalaze se skupine fontova. Nakon kreiranja projekta u toj mapi postoji grupa fontova pod nazivom Glyphonic-Halflings, te se tom skupinu fontova koristi Bootstrap okvir koji ujedno

zahtijeva da svi njegovi fontovi moraju biti u mapi za fontove. Sam programer može dodati nove fontove koji odgovaraju dizajnu stranice kakav je imao na umu.[8]

Mapa za skripte služi za pohranjivanje JavaScript datoteka, a već nakon kreiranja projekta ta mapa sadržava nekolicinu datoteka. Većina datoteka ima dvije verzije, jedna koja je normalna (original) i drugu, smanjena verzija koja ima nastavak min (kratica za minimum, minified, što znači da je riječ o minimiziranoj verziji). Sada će biti nabrojene te datoteke. Prve su Bootstrap datoteke i, kako samo ime kaže, to su datoteke za Bootstrap okvir. Sljedeće su jQuery datoteke, jQuery je JavaScript okvir koji općenito pojednostavnjuje rad, jer je u većini slučajeva lakši za uporabu od čistog JavaScripta, ali on će biti detaljnije objašnjen u drugom poglavlju. Uz samu jezgru jQuery okvira u MVC je automatski uključen priključak za validaciju, koji služi za jednostavnije provjeravanje podataka prije slanja na server. Sljedeća je datoteka modernizr,js koja služi za određivanje sposobnosti preglednika, a to radi s pomoću više testova koji se provode nad samim preglednikom, što znači da se ovaj alat ne oslanja na datoteke koje preglednici imaju i u kojima je nabrojeno što podržavaju (što je u redu jer pokatkad te datoteke nisu ažurne, odnosno ne odražavaju stvarno stanje preglednika). I zadnja skupina datoteka jesu respond.js datoteke. Respond.js je priključak za jQuery okvir koji služi kao alat za kreiranje web-stranica s interaktivnim (eng. responsive) sadržajem [8].

3.2.3. Kontroleri

Kontroleri su klase koje se brinu za zahtjeve koje šalju korisnici, odnosno oni sadržavaju određene akcije (metode) koje se izvršavaju kao odgovor na određene zahtjeve korisnika. Programer može sam napraviti cijelu kontroler klasu kako mu odgovora rabeći `IController`, ili može naslijediti neku od postojećih dviju apstraktnih klasa (`Controller`, `AsyncController`). Razlika između tih dviju klasa može se zaključiti i po samim nazivima pa tako `AsyncController` omogućuje kreiranje asinkronih akcija ako je to potrebno u aplikaciji [8].

U ASP.NET MVC verziji četiri uključen je način asinkronog rada koji se koristi klasom `Task` koja omogućuje pisanje asinkronih akcija. Za lakši rad s tom klasom rabe se dvije ključne riječi. `Await` (čekaj) je prva i označuje da metoda treba pričekati da se izvrši neki drugi dio koda ili neka metoda. `Async` je druga i njome se označuje da je riječ o asinkronoj metodi. Metode koje se koriste ovakvim načinom asinkronog rada vraćaju rezultat tipa `Task<ActionResult>` (Action Result, hrv. rezultat akcije predočuje jedan od mogućih rezultata koji vraćaju akcije, a biti će nabrojani poslije) [8].

Postoji mnogo rezultata koje mogu vratiti akcije, a neki od češće primjenjivanih bit će nabrojeni. `ViewResult` vraća pogled kao web-stranicu, postoji i za vraćanje parcijalnog pogleda. Sljedeća je `RedirectResult` koja služi kako bi se preusmjerilo na neku drugu akciju. `JsonResult` vraća rezultat u obliku JSON-a. `FileResault` vraća datoteku, a `HttpStatusCodeResult` vraća neki HTTP statusni kod kao rezultat (npr. 200 (OK)) [8].

3.2.4. Razor view engine

Primarna korist od Razor view enginea (hrv. Razor stroj za poglede, dalje samo: Razor) jest dodavanje C# (serverskog koda) na poglede, odnosno pisanje C# koda u kombinaciji sa samim HTML kodom. Cilj je ovoga da se može pisati čišći HTML i CSS kod te da se mogu raditi petlje, uvjeti i slične stvari u HTML dokumentu, što može biti vrlo korisno i ovakav način rada poprilično je sličan načinu rada PHP jezika koji se isto tako može uključiti u sam HTML dokument. Usto što se sam kod može pisati u HTML dokumentu moguće je i kreirati metode koje se zatim mogu rabiti na više mjesta (postoje tri vrste metoda koje mogu biti kreirane: HTML pomagači, Razor funkcije i Razor delegati). Naravno, te se metode mogu primjenjivati lokalno u pogledu ili čak globalno (mogu se rabiti u više pogleda), a globalnost se postiže tako da se sama metoda stavi u App_Code mapu [8].

3.2.5. HTML pomagači

Ovo su metode koje su unaprijed ugrađene u sam projekt i služe kako bi olakšale i ubrzale pisanje samog HTML koda. Neke od njih ovise o predlošcima koji se nalaze u samom projektu, a moguće je i kreiranje novih predloška (i novih HTML pomagača). Sada će biti nabrojeni i kratko opisani neki HTML pomagači. `ActionLink` omogućuje kreiranje linka, što je vrlo korisno jer je njime moguće napraviti link za koji ne moramo sami pisati logiku usmjeravanja. `TextBox`, `TextArea`, `RadioButton` itd. omogućuju kreiranje istoimenih (nisu svi) HTML elemenata. Editor omogućuje kreiranje komponente obrasca, odnosno input element, a, naravno, postoji i metoda za `<label>` element. `@Html.BeginForm()` služi za kreiranje početne oznake `<form>` elementa koja sadržava sve podatke koje bi sadržavala i početna HTML oznaka tog elementa. Pitanje je zašto se koristiti ovim metodama umjesto jednostavno napisati te HTML elemente, a neki od razloga jesu: pokatkad je kraće, otpornije je na promjene i jedno od bitnijih obilježja jest to da je lakše rabiti C# kod (varijable i slično) u kombinaciji sa spomenutim metodama nego sa samim HTML kodom [8].

3.2.6. MVC pogledi

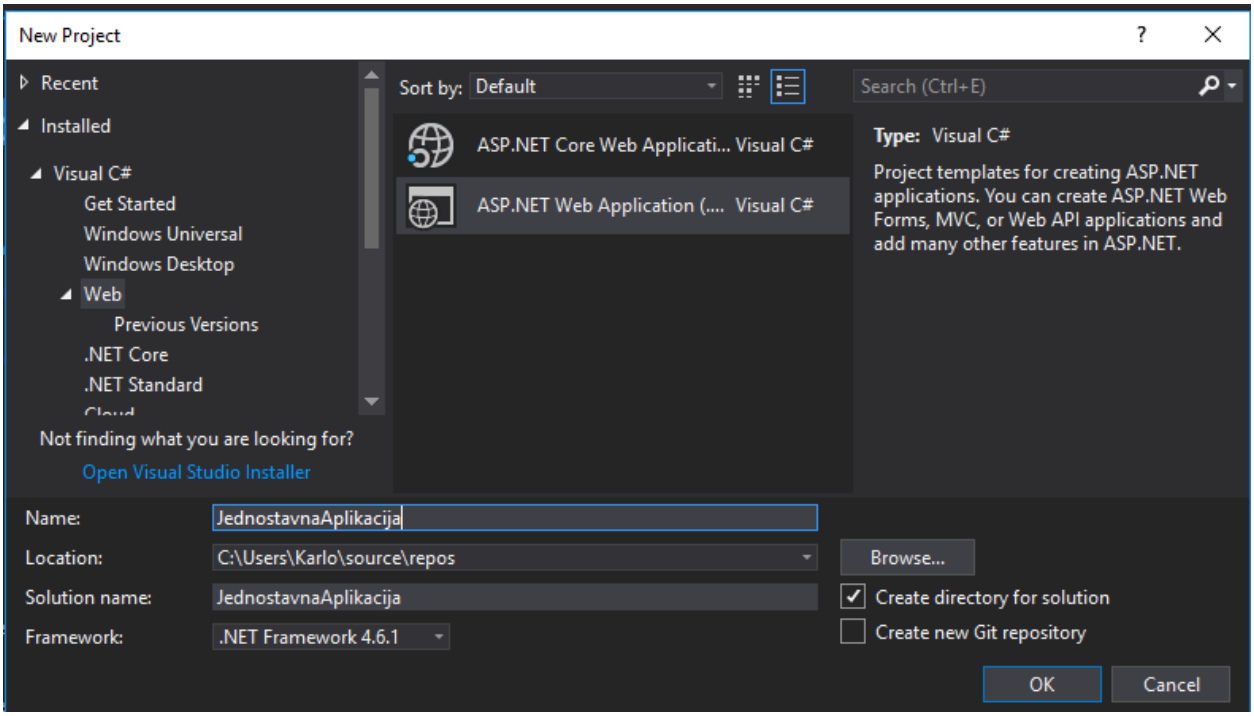
Pogledi su datoteke koje sadržavaju sve što će biti prikazano na ekranu i prenose zahtjeve kontrolerima na strani servera i pozivaju JavaScript metode za poslove koji se obavljaju na strani klijenta. Postoje tri vrste pogleda: shema (eng. layout), pogledi (eng. views) i parcijalni pogledi (učitavaju se bez sheme). Parcijalni pogledi služe za učahurivanje UI (eng. User Interface, korisničko sučelje) komponenta, što omogućuje da se taj parcijalni pogled rabi na više mjesta i tako smanji dupliciranje koda, ali, naravno, i parcijalni pogled može imati shemu. Neki primjeri za uporabu parcijalnih pogleda jesu: navigacijska traka, kontakt informacije na dnu stranice i zaglavlje stranice. Sve to omogućuje lakše pisanje vlastitih shema jer nije potrebno stalno pisati isti kod. Sheme su zapravo HTML dokumenti u kojima unaprijed postoje glavni dijelovi dokumenta zajedno s nekim HTML pomagačima. Naravno, svi će pogledi, ako nemaju definiranu shemu, imati zadanu pod nazivom _layout.cshtml. Dva su bitna dijela svake sheme. Prvi je tijelo kamo dolazi sam pogled, a drugi su sekcije koje zapravo predstavljaju određeni HTML kod (npr. zaglavlje) koji se može uključiti ako se to želi [8].

3.2.7. ASP identitet

Budući da je već više puta spomenut ASP.NET identitet (eng. identity), ovdje će biti ukratko opisan. ASP.NET identitet je Microsoftov sustav za autorizaciju korisnika preko nekog računa. On omogućuje prijavljivanje računima s drugih stranica kao što su: Facebook, Twitter, Google, Microsoftov račun itd. Također se taj sustav može rabiti sa svim ASP.NET okvirima i na svim uređajima, odnosno vrstama aplikacija. Sustav omogućuje selektivno pamćenje informacija, odnosno moguće je izabrati što je potrebno znati o korisniku (vrlo korisno u slučaju kada se korisnik koristi računom s neke druge stranice, jer možda svi podatci koje ta stranica pamti o korisniku nekoga ne zanimaju). Svi se podatci spremaju u bazu podataka te ih je lako kontrolirati i može se lako mijenjati sama shema baze podataka (preimenovati stupce u tablici ili samu tablicu, promijeniti primarni ključ itd.). Sam ASP.NET identitet previše je opširan te će zato biti nabrojene još samo neke njegove mogućnosti. Moguće je jedinično testiranje svih klasa koje se koriste ASP.NET identitet sustavom, moguće je kreiranje uloga i autorizacija pomoću uloga kao što su obični korisnik, moderator i administrator, autorizacija temeljena na zahtjevima itd. Sam ASP.NET identitet uključen je u ASP.NET okvire od Visual Studio verzije iz 2013. godine, ali se može preuzeti i iz NuGet rukovatelja paketima [9].

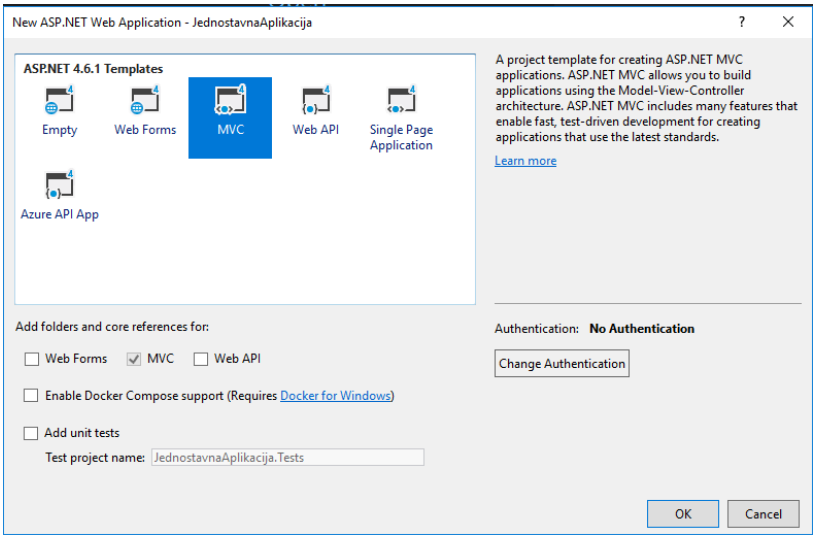
3.3. Jednostavna aplikacija

Jednostavna aplikacija koja će biti opisana u nastavku za temu ima knjižnicu. Postoje dvije glavne mogućnosti: prva je dodavanje novih članova i ona će primarno služiti kako bi se pokazao rad s obrascima, a druga je prikaz članova i knjiga u obliku tablica koja će služiti za prikaz rada s modelima. Prvo se u razvojnom okruženju treba kreirati projekt odabirom na datoteka (eng. file) i zatim novi projekt (eng. new project). Sljedeća slika prikazuje rezultat tog odabira.



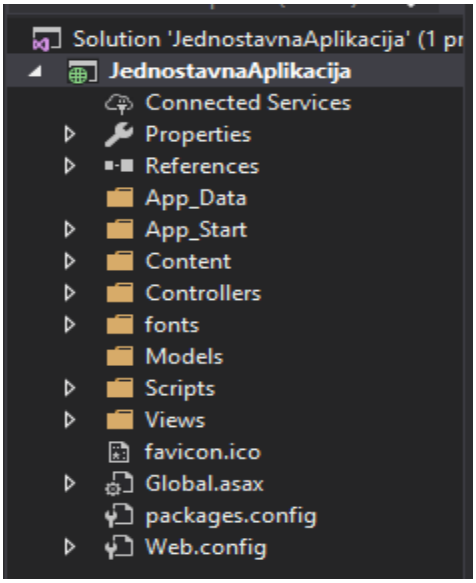
Slika 2. Prikaz izbornika za novi projekt u Viusal Studio razvojnom okruženju

Kao što se na slici vidi, da bi bila kreirana ASP.NET MVC aplikacija, mora se otići pod opciju za web te odabrati ASP.NET web-aplikacija. U donjem dijelu slike može se napisati željeno ime projekta, rješenja (eng. solution) i ondje će biti pohranjen projekt, te se može odabrati verzija .Net okvira koja će biti rabljena u projektu, što je vrlo bitno jer ima razlika između verzija. Usto, u desnom donjem kutu može se odabrati i to da se za projekt kreira novi Git repozitorij preko kojeg zatim programer može imati pristup povijesti promjena na projektu i šetati se po njima. Nakon klika na OK otvara se novi prozor koji je prikazan na slici 3.



Slika 3. Prikaz izbornika za odabir vrste aplikacije u Visaul Studio razvojnom okruženju

Ovaj je korak vrlo bitan pri izradi aplikacije. Ovdje se odabire vrsta aplikacije. U slučaju ove jednostavne aplikacije to će biti MVC aplikacija, a od ostalih odabira web-obrasci su već bili spomenuti, a popularan su odabir i web-API aplikacije. Osim toga, iako je odabran MVC, i dalje se mogu na donjem dijelu slike odabrati mape za web-obrasce i web API, a moguće je odabrati i jedinično testiranje. Često se kombinira MVC sa web-API opcijom. Na desnoj strani može se odabrati vrsta provjere korisnika (eng. Authentication), što može biti bez provjere, ili s provjerom preko baze podataka i provjerom preko nekih pružatelja takve usluge. Kada se pritisne OK, kreira se projekt koji ima strukturu koja je prikazana slici 4.



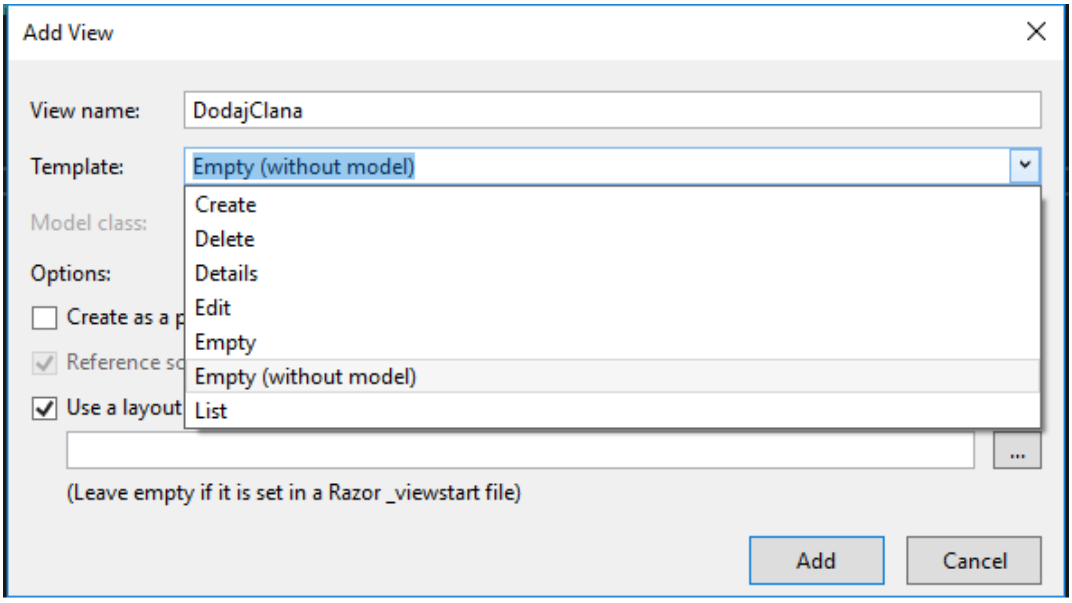
Slika 4. Prikaz datoteka i mapa dostupnih nakon kreiranja projekta

Kao što se vidi na slici, postoje zasebne mape za kontrolere, modele i poglede. Među njima su bitne mapa App_Start u kojoj se nalazi klasa za konfiguriranje ruta i klasa Global.asax.cs u kojoj se nalazi metoda koja se poziva kada aplikacija kreće s radom.

Što se tiče same aplikacije, prvo će biti opisana mogućnost kreiranja člana. Prije svega mora biti kreirana klasa koja će biti model za članove. U toj se klasi nalaze sva bitna svojstva koja su potrebna kako bi se opisalo člana (ime, prezime,…). Slijedi prikaz koda za neka od tih svojstava.

```
        [Required(ErrorMessage = "Morate unijeti Lozinku.")]
        [MinLength(8,   ErrorMessage   ="Unijeli   ste   premalo   znakova   za
lozinku.")]
        [MaxLength(50,ErrorMessage   ="Unijeli   ste   previše   znakova   za
lozinku.")]
        [DataType(DataType.Password)]
        public string Lozinka { get; set; }
        [Required(ErrorMessage = "Morate unijeti ponovljenu lozinku.")]
        [DataType(DataType.Password)]
        [Compare(nameof(Lozinka),ErrorMessage ="Lozinke nisu iste.")]
        [Display(Name ="Ponovite lozinku")]
        public string PonovljenaLozinka { get; set; }
```

U kodu se može primijetiti da postoje neke linije iznad svakog svojstva, a te linije služe za validaciju, što znači da će, kada se kreira obrazac za člana, te provjere biti primijenjene. Neke su od njih obvezan unos (eng. required) koji označuje da je polje obvezno, zatim postoji validacija za minimalnu i maksimalnu duljinu zapisa te tip podataka koji određuje kako će biti prikazano polje u obrascu, ali pokatkad i kojeg oblika to polje mora biti (npr. kod tipa e-mail mora postojati @ znak i nešto nakon njega, no to ne radi savršeno, ali je dovoljno dobro za uporabu). Usto, iznad drugog svojstva postoji i validacija usporedbom, odnosno provjerava se jesu li lozinke iste. Također se može primijetiti da postoji mogućnost unošenja vlastitih poruka u slučaju greške, što omogućuje da takva validacija bude iskorištena u bilo kojem jeziku. Još jedna zanimljiva validacija jesu regularni izrazi, tako da, ako validacija za email polje nije dovoljno dobra, može se kreirati regularni izraz koji će to provjeravati. S tim validacijama i s mogućnošću da se zada ime svojstva za prikaz omogućuje se automatsko kreiranje obrasca. Same po sebi te su validacije u redu, ali definitivno ne pokrivaju sve, pa se za neke situacije i dalje treba rabiti JavaScript. Uz ova u kodu prikazana svojstva, još postoje ime, prezime i e-mail. Model knjiga ima svojstva naslov, ime i prezime autora. Kontroler za ove mogućnosti zove se knjižnica kontroler i u njemu se nalaze akcije koje se pozivaju kako bi se te mogućnosti pokrenule. Metoda za kreiranje člana zove se dodaj člana i vraća pogled. Sljedeća slika prikazuje kreiranje novog pogleda.



Slika 5. Prikaz izbornika za kreiranje novog pogleda

Ako se iz metode pokrene kreiranje pogleda, ime tog pogleda bit će isto kao i ime metode. Na slici se vide predlošci za izradu pogleda, kao što je kreiranje, brisanje, lista itd. U slučaju kreiranja člana bit će primijenjen predložak za kreiranje te će biti odabrana klasa član za model pogleda. Svi pogledi kojima ne specificiramo shemu (eng. layout) imaju zajednički shemu koja je automatski napravljena. Ona se sastoji od navigacije, mjesta za sadržaj pogleda, naslova aplikacije, podnožja, itd.

Nakon što je kreiran pogled obrazac je već spreman za uporabu, ali su potrebne neke male promjene, kao što je uklanjanje nepotrebnog linka i preimenovanje gumba. Ispod je naveden kod za jedno polje obrasca.

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

Prvi je naveden kod za validaciju, a zatim kontejner s klasom za obrazac unutar kojeg se nalazi C# kod unutar Razora koji služi za kreiranje oznake (eng. label), nakon čega slijedi još jedan kontejner u kojemu se nalazi samo polje za unos i validacija. Vidljivo je da se koristi Razor stroj

po tome što linije počinju znakom `@` i nakon toga slijedi C# kod. Model koji se spominje jest, naravno, model za člana. Nakon što je obrazac popunjen i poslan, on se šalje istoj metodi, ali drugom obliku te metode koji za parametar prima model i iznad te metode mora biti linija koja označuje da se ta metoda uporabljuje za POST metodu zahtjeva, odnosno za primanje zahtjeva od korisnika. Ta metoda u ovoj jednostavnoj aplikaciji samo preusmjeruje korisnika na početnu stranicu.

Sljedeća mogućnost jest prikaz u obliku tablice. Da bi se uopće mogli prikazati podatci različitih modela u istom pogledu, prvo je potrebno napraviti novi model koji će u sebi imati sve podatke koji se žele prikazati, a to su lista članova i lista knjiga (postoje i drugi načini za slanje više modela pogledu, ali ovaj je najjednostavniji i vrlo pouzdan). U ovom slučaju, ako se odabere lista iz predloška za izradu pogleda, neće biti kreiran nikakav kod (jer u modelu postoje dvije liste), ali ono što se može napraviti, jest to da se zasebno kreira pogled s modelom člana i s modelom knjiga i zatim kopira kod u zajednički pogled (naravno, potrebne su promjene kako bi to radilo, ali one nisu velike). U nastavku je naveden dio koda za prikaz u obliku tablice.

```
<th>
   @Html.DisplayNameFor(model => model.knjige.First().AutorPrezime)
</th>
@foreach (var item in Model.knjige)
    {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Naslov)
            </td>
```

Naravno, iznad navedenog koda sama je oznaka za tablicu i to nije prikazana cijela tablica. Vidljivo je u kodu da se za glavu tablice uporabljuje naredba koja dohvaća ime za prikaz od nekog svojstva, a u petlji ispod prikazuju se sami podatci.

Zadnja stvar koja će biti ukratko spomenuta jest usmjerivanje koje se nalazi u klasi RouteConfig.cs. U nastavku je prikazan kod iz te klase.

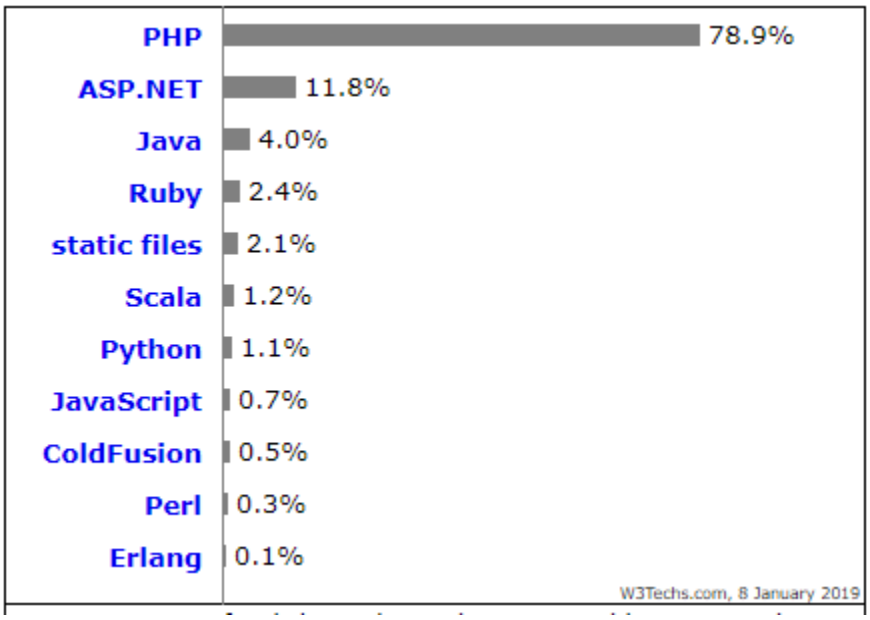
```
routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
routes.MapRoute(
  name: "Default",
  url: "{controller}/{action}/{id}",
  defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional }
  );
```

Prva je linija za ignoriranje ruta, a dalje se nalazi naredba za kreiranje mape usmjerivanja. Trenutačno postoji samo jedna takva naredba i to je ona koja je kreirana pri kreiranju projekta i

njezin oblik URL-a jest da prvo ide kontroler, zatim akcija i opcionalni parametar `id`. Parametar `id` je opcionalni jer u liniji ispod piše da je zadana (eng. default) vrijednost home kontroler, indeks akcija i na posljetku je opcionalni parametar (to je `id`). Nove mape usmjeravanja mogu biti kreirane ako za to postoji potreba, ali u većini slučajeva ne postoji. Ono što je bitno jest da sve nove mape stavimo iznad postojeće jer se podudaranje URL-a traži odozgo prema dolje.

4. Usporedba C# jezika s drugim programskim jezicima

Prije samih usporedbi biti će prikazano trenutačno stanje popularnijih jezika za programiranje na strani servera na tržištu (koliko se uporabljuju i za kakve stranice).

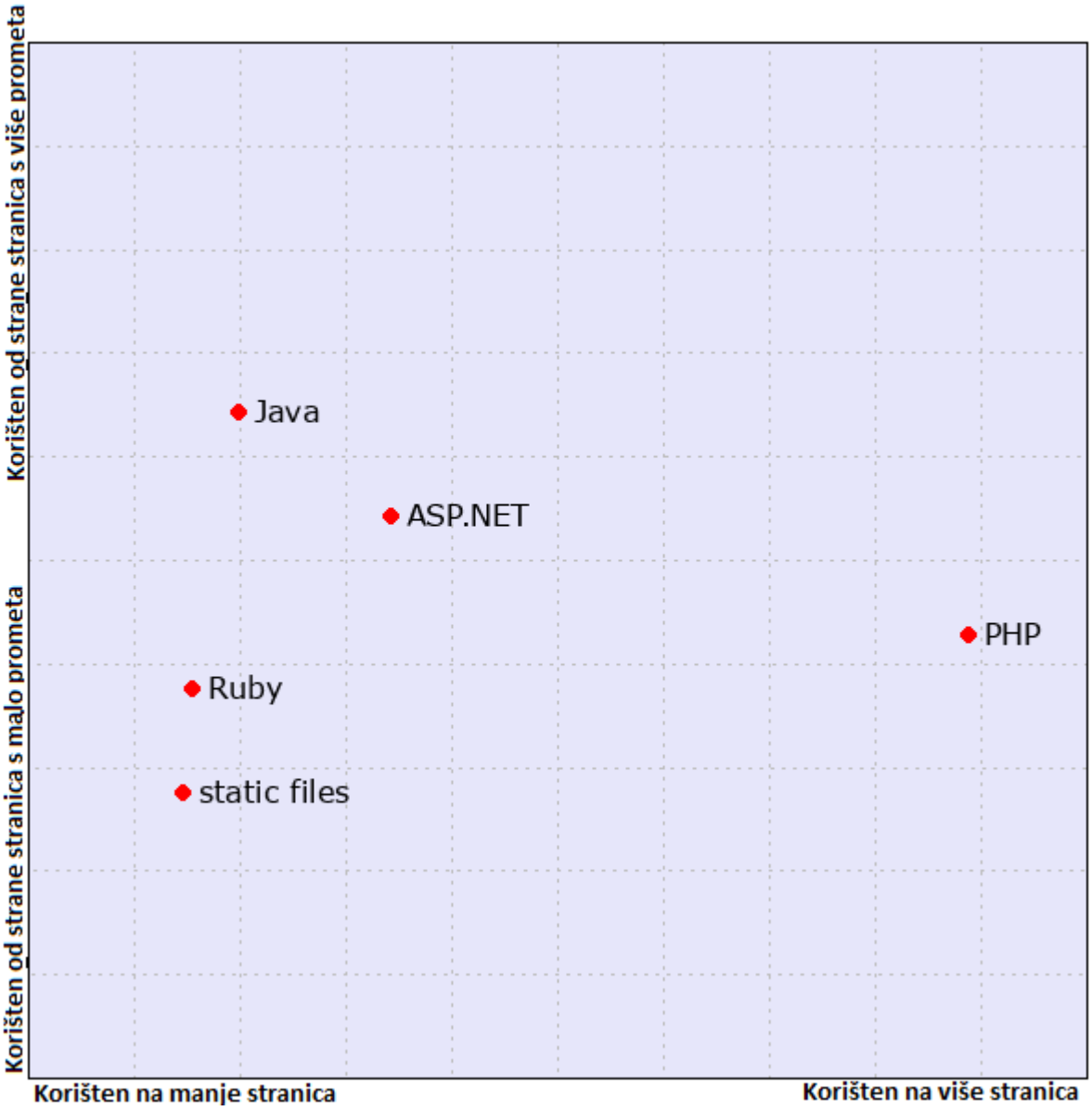


Slika 6. Uporaba jezika na strani servera na stranicama za godinu 2019. (izvor: https://w3techs.com/technologies/overview/programming_language/all , 2019)

Prethodni graf prikazuje uporabu nekog jezika na web-stranicama za siječanj 2019. godine. Graf prikazuje da se još uvijek na većini stranica na internetu uporabljuje PHP kao primarni jezik za programiranje na strani klijenta (naravno, neka web-aplikacija može biti napravljena s više različitih jezika na strani servera). ASP.NET je odmah ispod PHP jezika, ali je razlika između dvaju jezika velika. Naravno na grafu nisu prikazani svi jezici, nego samo oni koji se najviše uporabljuju, uz napomenu da su nakon prvih sedam jezika svi ostali ispod jedan posto. PHP je relativno stariji jezik i među prvima u ulozi jezika za programiranje na strani servera i njegova je glavna uloga programiranje na webu (za razliku od nekih drugih jezika kao C# i Java koji se rabe i za desktop aplikacije). Tijekom godina PHP je nakupio veliku zajednicu i ima mnogo prinosnika koji izrađuju alate i funkcionalnosti.

Sljedeći graf prikazuje pozicije na tržištu različitih programskih jezika na strani klijenta i sada će ukratko biti razjašnjene te pozicije. Što je neki jezik desnije na grafu, to se više uporabljuje i zato je PHP koji se najviše upotrebljava najdesnije na grafu. Što je jezik na višoj poziciji na grafu, to znači da se rabi na stranicama koje imaju mnogo prometa i na kojima treba brzo procesirati više

zahtjeva. Na najvišem je položaju Java programski jezik, a odmah su ispod njega ASP.NET i PHP. Na grafu se uočava da nema veće razlike u visini između ASP.NET-a i PHP jezika, već je najveća razlika u korištenosti samog jezika.



Slika 7. Marketinška pozicija jezika na strani servera za godinu 2019. (izvor: https://w3techs.com/technologies/market/programming_language, 2019)

4.1. PHP i C#/ASP.NET

Sam PHP jezik mnogo je stariji jer je ranije bio razvijen i prije se počeo upotrebljavati kao programski jezik na strani servera. Činjenica da je stariji i neko je vrijeme bio praktički jedini jezik koji se upotrebljavao za tu svrhu omogućila je PHP jeziku da razvije veliku zajednicu aktivnih korisnika. To što je bio otvoren kod, tako da je bilo tko mogao na njega dodavati nove ili mijenjati postojeće mogućnosti uvelike je pridonijelo njegovu razvoju, ali isto tako to je donekle razlog postojanja mnogih funkcionalnosti koje nisu savršene i mnogo dokumentacije koja jednostavno više ne vrijedi, a samim time i mnogo vodiča koji su zastarjeli. S druge strane. C# i ASP.NET u vlasništvu su tvrtke Microsoft te ih ona razvija i nadograđuje. To ne znači da za taj jezik postoji mnogo manje alata i opcija, nego samo da zajednica ima manji utjecaj na sam jezik i njegov razvoj. PHP također još uvijek ima mnogo veću zajednicu, a to znači da je lakše doći do odgovora na pitanja nego kod C# jezika [16] [17] [18].

Što se tiče učenja jezika od početka, PHP je definitivno jednostavniji od dvaju jezika i mnogo lakši za takvo učenje (relativno je sličan jeziku C i relativno je intuitivan te pokatkad čak može raditi i s nekim greškama i manje je strog u tom smislu, ali to, naravno, može biti i loša stvar). Jezik C# mnogo je sličniji Java jeziku nego bilo kojem od jezika C i sam je po sebi kompliciraniji da bi se naučio ako programer nema iskustva s Java jezikom. C# jezik najčešće se uporabljuje u Visual Studio razvojnom okruženju koje je osmišljeno za izradu Microsoftovih proizvoda i podupire više jezika, uključujući i C#. Ovo okruženje ima vrlo detaljan sustav za otkrivanje grešaka i odlično automatsko završavanje linija i vrlo često nudi rješenja za greške, detaljan opis grešaka i dokumentaciju vezanu za njih. Također ovaj sustav omogućuje izvršavanja koda od određene linije pa čak i preskakanje dijelova koda i prikaz vrijednosti varijabli u nekom trenutku. PHP ima mnogo alata koji ubrzavaju sam proces razvoja, a u to je naravno uključeno i dobro otkrivanje grešaka [16] [17] [18].

Kao što je već bilo spomenuto, PHP je besplatni i otvoreni (eng. Open source) jezik, dok je C# u vlasništvu tvrtke Microsoft i isto tako je besplatan, ali se samo okružje koje se najčešće uporabljuje s njim i neki alati plaćaju (naravno, okružje ima i besplatnu verziju, ali za tvrtke koje rabe jezik C# uobičajeno je potrebna plaćena verzija). Usto, tehnički gledano, tvrtka koja želi rabiti C# mora kupiti Windows operacijski sustav (može se izbjeći, ali nije praktično), te Microsoftov server i server za bazu, što ispada dosta skupo. S druge strane, PHP se može jednostavno upotrebljavati i s Linux operacijskim sustavom i Apache serverom koji su besplatni. Što se tiče operacijskih sustava, PHP ih podržava mnogo, dok je C# dugo bio ograničen na

Windows operacijski sustav. S novijim verzijama ASP.NET MVC-a podržani su i Linux i Mac operacijski sustavi [16] [17] [18].

Glede sigurnosti, PHP je slabiji jer je bio kreiran u vrijeme kada se sigurnost još nije smatrala bitna u smislu da tada još nisu postajale prijetnje koje danas postoje. PHP ima fokus na stranice koje su više prilagođene korisnicima i potrebe korisnika na prvom su im mjestu. S druge strane, C# ima mnogo veći fokus na sigurnost podataka i stranice općenito i na same funkcionalnosti stranice, a manje na potrebe samih korisnika [16] [17] [18].

Oba jezika mogu služiti za izradu većih i manjih web-aplikacija te, što se brzine tiče, ne postoji pretjerano velika razlika, odnosno na brzinu mnogo više utječu same mogućnosti korisnika kao što je brzina prijenosa i koliko se podataka odjednom može prenijeti. PHP je dobar za izradu CRM (eng. Customer Relationship Management, upravljanje odnosa s klijentima) stranica koje općenito služe za upravljanje odnosa s klijentima i njihovim podatcima, stranica za manje tvrtke, razvojnih tvrtki (eng. Startup), stranica za naplaćivanje. Jezik C# koristan je pak za izradu stranica za veća poduzeća, intraneta (privatna računalna mreža) i ERP aplikacija (koje se rabe unutar nekog poduzeća), što znači da je C# bolje upotrebljavati tamo gdje je bitnija sigurnost i nema velike potrebe za brigu o potrebama korisnika (jer će korisnici biti unutar poduzeća i bit će podučeni kako raditi sa programom) i zato je za C# savršeno mjesto unutar organizacije gdje je jako bitno da informacije ne procure i da netko ne probije sigurnost poduzeća i ukrade osjetljive podatke (naravno i sam PHP jezik ima opcije za sigurnost) [16] [17] [18].

Sve u svemu, oba jezika imaju prednosti i nedostatke, trenutačno se PHP mnogo više upotrebljava, ali i C# i ASP.NET imaju svoju ulogu u svijetu razvoja web-aplikacija. Koji se jezik upotrebljava, ovisi o prijašnjem znanju programera, potrebama klijenta, roku do kada treba isporučiti aplikaciju i proračunu. Ako je programer naviknut na rad u javi i na stroži jezik koji ne dopušta nikakve nepravilnosti u kodu, možda je čak i lakše savladati C#, ali, ako je riječ o osobi koja se prvi put susreće s razvojem web-aplikacija, PHP je definitivno bolji izbor [16] [17] [18].

4.2. Node.js vs. C#/ASP.NET

Node.js rabi se za programiranje na strani servera i kao primarni jezik ima JavaScript (mogu se upotrebljavati i neke inačice JavaScript jezika kao što je Microsoft TypeScript). ASP.NET može koristiti više jezika, ali je to najčešće C#. Naravno, kao što je već bilo spomenuto, C# ima dobru podršku za lociranje i ispravljanje grešaka te automatsko dovršavanje, pa čak i kreiranje dijelova koda. S druge strane, Node.js koristi se JavaScriptom koji sam po sebi nije tako lagan za ispravljanje zbog manje čvrstih pravila pri kodiranju (npr. ne mora se napisati tip podataka koji će poprimiti varijabla), ali postoje alati koji provjeravaju tipove varijabli pa se mogu lakše pronaći takve greške, a i neki jezici koji se mogu prevesti u JavaScript i vrlo su slični tom jeziku i bolje otkrivaju greške [19] [20].

Po krivulji učenja C# jezik je dosta težak jer je objektno orijentiran (što znači da treba dobro poznavati koncept klasa), dok se Node.js koristi JavaScriptom koji je mnogo lakši za svladavanje, ali, da bi programer svladao Node.js, kad-tad će morati naučiti asinkrono programiranje koje isto tako nije najjednostavnije. Baš zbog svoje strogoće pisanja koda C# je otporniji na greške i iznimke jer ih veliku većinu prepozna te ima i dobar sustav za lovljenje grešaka (eng. try catch). JavaScript jeziku samom po sebi nedostaje ta strogoća pisanja koda, a to se još više ističe kada je u pitanju asinkrono programiranje [19] [20].

Što se tiče same arhitekture i načina kreiranja aplikacije, programiranje u jeziku Node.js svodi se na pisanje mnogo malih komponenti, što omogućuje vrlo fleksibilan pristup prema rješenju problema. Imajući na umu da je riječ o takvoj arhitekturi, moglo bi se doći do zaključka da se često mora pisati isti kod na više mjesta, ali, nasreću, to nije slučaj jer Node.js nudi sve što je potrebno da bi se to izbjeglo u svojim bibliotekama, ili u bibliotekama trećih strana (eng. third party). S druge strane, ako pogledamo APS.NET-ov MVC uzorak, on ima više konvencija od konfiguracija, što znači da unaprijed ima zadanu strukturu projekta i pravila nazivanja određenih datoteka. Takav pristup zahtijeva od programera da se drži tih pravila, ali dalje od toga programer može raditi što želi. Naravno, problemi se pojavljuju kada takav način rada nije dobar za zadatak koji se treba napraviti i tada programer mora prekršiti pravila, što znači da sada više ništa nije automatski povezano ni konfigurirano, pa programer mora sve to sam napraviti ili tražiti rješenja od treće strane. Također takav način nije potreban za manje aplikacije i čini se pretjeran [19] [20].

Što se tiče same performanse, oba su jezika bolja u nečemu. Brzini Node.js jezika pridonosi to što je sinkronizacija između strane servera i strane klijenta brža budući da je riječ o istom programskom jeziku na objema stranama. Node.js može vrlo brzo obrađivati zahtjeve i općenito

je lagan i, budući da implementira asinkroni način rada, savršen je za česta ažuriranja. S druge strane, iako C# ima podršku za asinkrono programiranje, općenito je sporiji zbog većeg fokusa na sigurnost i općenito strože sintakse, ali u slučaju većih i kompleksnijih izračuna C# uvijek će pobijediti Node.js u performansi [19] [20].

Budući da je Node.js jako lagan i brz, savršen je za izradu aplikacija koje rade u stvarnom vremenu (eng. real-time application) i koje su temeljene na događajima. Zato se često uporabljuje pri izradi aplikacija za dopisivanje i videokonferencije i ostale takve aplikacije koje zahtijevaju često ažuriranje. Naravno, ako je riječ o aplikacijama koje zahtijevaju kompleksne izračune, C# je bolja opcija (aplikacije unutar poduzeća koje zahtijevaju kreiranje izvještaja i računanja mogućih ishoda itd.) [19] [20].

Što se tiče troška, Node.js je općenito jeftiniji jer je otvoreni kod i bilo tko ga može rabiti i nadograđivati, a i većina je tehnologija koje se primjenjuju u kombinaciji s njim besplatna. S druge strane, za rad u C# potrebno je kupiti Microsoftov server, server za bazu itd. [19] [20].

Uz sve to postoji još jedna prednost u Node.js jeziku koju vrijedi spomenuti, a to je činjenica da zato što se piše u JavaScript jeziku mnogi programeri koji su radili na strani klijenta mogu relativno jednostavno naučiti raditi na strani servera koristeći se jezikom Node.js, a i općenita komunikacija između programera s obiju strane postaje jednostavnija [19] [20].

Koji jezik odabrati, ovisi o tome koji zadatak treba riješiti. Ako je potrebno brzo obrađivanje zahtjeva koji ne zahtijevaju teže izračune, Node.js je vjerojatno bolji izbor i jeftiniji, ali, ako aplikacija zahtijeva veću sigurnost i teže izračune, bolja je opcija C# [19] [20].

4.3. Python vs. C#/ASP.NET

Ako gledamo općenito, oba su jezika objektno orijentirana (C# strogo, dok u Pythonu se još uvijek može pisati proceduralni kod). Najveća razlika između dvaju jezika jest u tome to što je Python dinamički interpreterski jezik, a C# statični kompajlerski jezik. Oba se jezika mogu rabiti za razvoj različitih vrsta aplikacija (od desktop-aplikacija do web-aplikacija) [21] [22] [23].

Python je otvoreni kod, što znači da je besplatan i da ga bilo tko može nadograditi i kreirati alate za jezik. Za sam razvoj web-aplikacija Python ima više web-okvira koji se mogu iskoristiti za taj zadatak, a jedan od popularnijih je Django. S druge strane, C# je u vlasništvu tvrtke Microsoft i treba platiti Microsoftov server i server za bazu kako bi se mogla objaviti web-aplikacija [21] [22] [23].

Što se tiče podrške tijekom programiranja, Python ima veliku aktivnu zajednicu od koje se mogu tražiti odgovori. C# isto tako ima poprilično veliku zajednicu, ali, usto, postoje i neke plaćene

opcije koje omogućuju programeru da dobije pomoć od nekoga tko radi izravno za tvrku Microsoft. [21] [22] [23]

Krivulja učenja za Python lakša je od one za C#. Python ne zahtjeva mnogo prijašnjega znanja kako bi se počelo programirati u njemu i sam programer može postupno nadograđivati svoje znanje kako nastaju nove prepreke koje zahtijevaju neke druge mogućnosti. S druge strane, C# zahtijeva da programer odmah zna dosta o samom jeziku prije nego što može početi programirati (klase, metode itd.) [21] [22] [23].

Što se tiče razvoja općenito, razvoj u Python jeziku i Python okvirima brži je i jednostavniji i jeftiniji. C# je stroži jezik i zahtijeva gradnju aplikacije koja uzima vremena, ali istodobno osigurava da se ne pojave greške. Oba jezika imaju podršku za više operacijskih sustava, ali Python je općenito jezik koji se može primjenjivati s drugim operacijskim sustavima, dok je C# relativno nedavno počeo podržavati druge operacijske sustave [21][22][23].

Python je općenito bolji izbor ako se s radom kreće od početka i treba brzo razviti neki proizvod i općenito je lakši za svladavanje, ali, ako tvrtka ima resurse da plati sve što je potrebno za razvoj u C# jeziku, on je vrlo pouzdan i siguran jezik s mnogo kvalitetnih alata na raspolaganju [21] [22] [23].

5. Okviri i biblioteke za razvoj korisničke strane web-aplikacije

U ovom će poglavlju biti opisana tri popularnija okvira za rad na korisničkoj strani aplikacije i sam jQuery koji je na neki način bio među prvim okvirima. Ti okviri će biti uspoređeni.

5.1. jQuery

jQuery je JavaScript biblioteka koja omogućuje jednostavnije izvođenje JavaScript zadataka i nema potrebe za fallback (kod koji se izvršava ako da preglednik ne podržava određenu mogućnost) kodom. Podržava ga većina preglednika (svi koji se najčešće uporabljuju). Iako tehnički nije okvir, vrlo je koristan i bit će uspoređen s okvirima te će zato biti kratko opisan (i neki drugi u ovom poglavlju neće biti okviri). Sam jQuery uključuje se u svaku stranicu koja se njime koristi, a riječ je o jednoj JavaScript datoteci, što znači da je vrlo malen okvir (bitno je specificirati verziju kako se preglednik ne bi koristio nekom starom ili novijom verzijom od željene). jQuery omogućuje jednostavnije selektiranje elemenata te također ima mnogo više mogućnosti za mijenjanje i rad s tim elementima. Selektori u jQueryju vrlo su slični CSS selektorima po tome što se može dohvatiti neki element s pomoću njegovog ID-a, klase, roditelja itd. Moguće je šetati po cijelom dokumentu u dubinu sve dok se ne dođe do željenog elementa. Samim time što je moguća selekcija s pomoću dokumenta znači da je moguće jednostavno dohvatiti elemente koji su bili naknadno dodani u sam dokument (npr. dinamički kreiran obrazac s pomoću JavaScripta). Sam selektor je oblika `$('selektor')`, što je vrlo lako zapamtiti, te se može spremiti u varijablu ili je moguće i odmah iskoristiti neku od jQuery metoda kako bi se promijenio taj element. jQuery također pojednostavnjuje kreiranje i brisanje samih elemenata te dodavanje animacija. Samo kreiranje i mijenjanje elemenata jest jednostavnije jer se može proći kroz cijelu skupinu elemenata u jednoj petlji. Također omogućuje dodavanje rukovatelja događaja na elemente ili samo jedan element (npr. može se staviti rukovatelj događaja na sve `<input>` elemente koji provjerava postoje li neki specijalni znakovi koji nisu dopušteni), i to u samo jednoj liniji koda. Sam jQuery može mijenjati atribute, stilove i klase elemenata, a selektori omogućuju da se promijeni neko svojstvo svim elementima koji su obuhvaćeni tim selektorom i to sve je isto u samo jednoj liniji. Usto, moguće je mijenjati više obilježja ili nešto slično jer jQuery omogućuje ulančavanje metoda, što znači da je moguće pozvati više metoda nad istim elementom ili elementima tako da se metode odvoje točkom. Za

ljude koji su naviknuli na JavaScript, jQuery je savršen jer je vrlo sličan i ne mijenja ono što JavaScript radi, već znatno pojednostavnjuje rad na strani servera, a i vrlo se lagano kombinira s JavaScriptom, štoviše, teško je koristiti samo jedno. jQuery znatno pojednostavnjuje AJAX (eng. Asynchronous JavaScript and XML, što se prevodi kao asinkroni JavaScript i XML) zahtjeve, te ih je moguće napisati sa samo nekoliko osnovnih informacija (kao URL i podatci koji se šalju). Prije je bilo spomenuto da ne treba fallback kod i to je tako jer se jQuery sam brine za različitosti između različitih preglednika (uvijek traži najbolji način na koji se zadatak može izvršiti u nekom pregledniku). Verzija 2.0 maknula je neke podrške za starije verzije preglednika kako bi ubrzala rad same skripte [10].

5.2. AngularJS

AngularJS je jedan od najpopularnijih, ako ne i najpopularniji okvir na strani klijenta, jednim dijelom zbog svojih mogućnosti, a dijelom zato što ga je razvila popularna tvrtka Google. Prvo će biti opisane neke od mogućnosti tog okvira koje ga čine popularnim. Okvir je moguće rabiti u samom HTML kodu, što znači da proširuje sam vokabular HTML-a te ga može napraviti dinamičnim. Sljedeća mogućnost jest dvosmjerno vezanje podataka (eng. Two way data-binding) koja pruža to da se ne treba brinuti o ažuriranju pogleda nakon promjene nad podatcima, a ni obrnuto jer se o tome brine okvir, a to znači da su sve promjene nad podatcima automatski vidljive na pogledu i automatski prenesene s pogleda na podatke. Sljedeća su mogućnost jednostavni i lako čitljivi kontroleri. Modeli tog okvira napisani su u samom JavaScriptu, što omogućuje lakše testiranje, mijenjanje i ponovnu uporabu dijelova koda. Sljedeća mogućnost jest duboko povezivanje (eng. deep linking), što je zapravo kreiranje linkova koji pokazuju na određeni dio stranice, odnosno na neku stranicu koja je dublje u hijerarhiji stranice (nije početna stranica, npr. link na neki specifičan proizvod na eBay stranici). Ovo također omogućuje korisnicima da zapamte određene dijelova aplikacije s pomoću bookmark mogućnosti preglednika. Sljedeća mogućnost jest validacija obrazaca, naime, taj okvir omogućuje pisanje validacijskih pravila a da se ne pišu u samom JavaScriptu i time smanjuje količinu potrebnog koda i pojednostavnjuje kontrolu na strani klijenta. Taj okvir također omogućuje jednostavniji rad sa serverskom stranom te lakši rad sa asinkronim dohvaćanjem podataka. Sljedeća su mogućnost direktive koje služe za ekspandiranje HTML koda, tj. one govore samom okviru da tijekom kompilacije tom DOM (eng. Document Object Model, dokument objekt model) objektu treba dodati neko određeno ponašanje (postoje već mnoge ugrađene, ali mogu se kreirati i nove). Spomenute direktive također služe kao alata za ponovno

iskorištenje koda jer se njima mogu opisati razne DOM strukture, CSS kod i ponašanja objekata. Okvir također nudi alate za lokalizaciju aplikacije kako bi bila dostupna u različitim jezicima i prijateljska prema različitim kulturama. Također se kao jedno od glavnih načela primjenjuje ubrizgivanje ovisnosti (eng. dependency injection) koje omogućuje da se objektima daju potrebne ovisnosti. Okvir omogućuje i testiranje svojeg koda te za to ima i neke svoje alate [11].

Naravno, svaki okvir ima i neke nedostatke koji će sada biti ukratko nabrojeni. Prvi je nedostatak poprilično strma krivulja učenja, što može odbiti moguće korisnike okvira (čak i za mali projekt zahtijeva duboko znanje okvira, koristi se mnogim specijalnim znakovima koje je teško zapamtiti, često se pojavljuju skrivene greške koje je teško otkriti jer sam okvir ne pokazuje na njih). Sljedeći nedostatak, moglo bi se reći, jest u tome da, ako se radi u tom okviru, onda sve treba raditi u njemu, a to znači da se sav kod na strani klijenta mora pisati koristeći se logikom AngularJS okvira (direktive, kod, validaciju, povezivanje sa serverom, poslovna logika itd.), što znači da nema nekog izbora korištenja nečega drugoga za razvoj na strani klijenta uz sam okvir. Taj okvir također često nema savršenu kompatibilnost unatraške te zahtijeva mnogo održavanja u tom smislu. Okvir ima i određena pravila i načine na koji aplikacija mora biti strukturirana (specifična arhitektura), ali to nije toliko čvrsto određeno, pa omogućuje neku slobodu, ali za neke korisnike to nije dovoljno. Iako ima svoje alate za testiranje, samo testiranje nije optimalno i često je nepraktično, a za neke stvari i nepostojeće. Njihov alat za testiranje koristi se testiranjem od kraja do kraja (eng. end-to-end) koje je poprilično sporo. Jedinično testiranje često ne funkcionira kako bi trebalo ili nije uopće podržano za neke dijelove aplikacije. Sve u svemu, AngularJS je okvir koji zahtijeva mnogo uloženog vremena, koji ograničuje svojeg korisnika na uporabu određenog načina kreiranja web-aplikacija, a zauzvrat nudi mnogo korisnih funkcionalnosti i mogućnosti, ali koje pokatkad mogu zatrpati samog korisnika [12].

5.3. React

React je JavaScript biblioteka koja primarno služi za izgradnju korisničkih sučelja te nastoji ostvariti što bržu komunikaciju s korisnikom, odnosno da što brže i efikasnije odgovora na zahtjeve korisnika. U nedavnim godinama ta je biblioteka postala jako popularna baš zbog tog novog načina učitavanja stranica, što stvara dinamično okružje i omogućuje vrlo brzu komunikaciju s korisnicima. Poslije je ista tvrtka kreirala React Native koji je okvir za razvoj na mobitelu, podržava i Android i iOS. U nastavku će biti nabrojena neka glavne obilježja koje ovaj

okvir/biblioteka ima. Prvo je virtualni DOM, što uvelike ubrzava reakciju na rad korisnika jer reagira na svaku malu promjenu i istodobno ne utječe ni na jedan dio sučelja koji nema veze s tim zahtjevom korisnika. Stvarni DOM kako bi reagirao na zahtjev korisnika mora ažurirati cijeli dokument. React također oslobađa programere od zadatka da spajaju DOM s funkcionalnostima na sučelju jer su elementi automatski već spojeni. Sljedeća velika prednost jest mogućnost ponovnog iskorištenja komponenti i izolirane komponente, što može uvelike uštedjeti vrijeme. Ponovno iskorištenje znači da se neka komponenta može koristiti više puta u istom ili nekom drugom projektu, a izoliranost komponenti znači da, ako je potrebno ažurirati neku komponentu, ta promjena neće utjecati na druge komponente. Sljedeće je vezanje podataka prema dolje (eng. Downward data binding) koje osigurava stabilnost koda. Vezanje podataka prema dolje znači da, u nekoj hijerarhiji elemenata djeca ne mogu utjecati na roditelje, tako da, dođe li do promjene, jedino što će se promijeniti jesu komponente koje su na toj razini ili niže. Sljedeće je to što je ova biblioteka otvoreni kod te se zbog toga brže razvija jer ima mnogo programera koji podupiru i poboljšavaju rad same biblioteke, što uključuje čak neke bitnije funkcionalnosti, ali i dodatne alate. Sljedeće je Redux koji ne mora biti rabljen uz ovaj okvir i može se upotrebljavati i s drugim okvirima, ali se jako često koristi s React okvirom. Redux služi za lakše spremanje i rad s komponentama, što je pogotovo korisno ako je riječ o velikoj aplikaciji. To radi tako da omogućuje komponentama izravan pristup centraliziranom objektu za spremanje podataka. Tako dvije komponente koje se nalaze na različitim dijelovima stabla, a imaju isto stanje, mogu izravno pristupiti tom stanju a da ne prolaze kroz više komponenta. Od alata koji su korisni za lakši rad s ovim okvirom ističu se oni koji se vežu uz preglednike, odnosno omogućuju pregled stanja i virtualnog DOM drva [13].

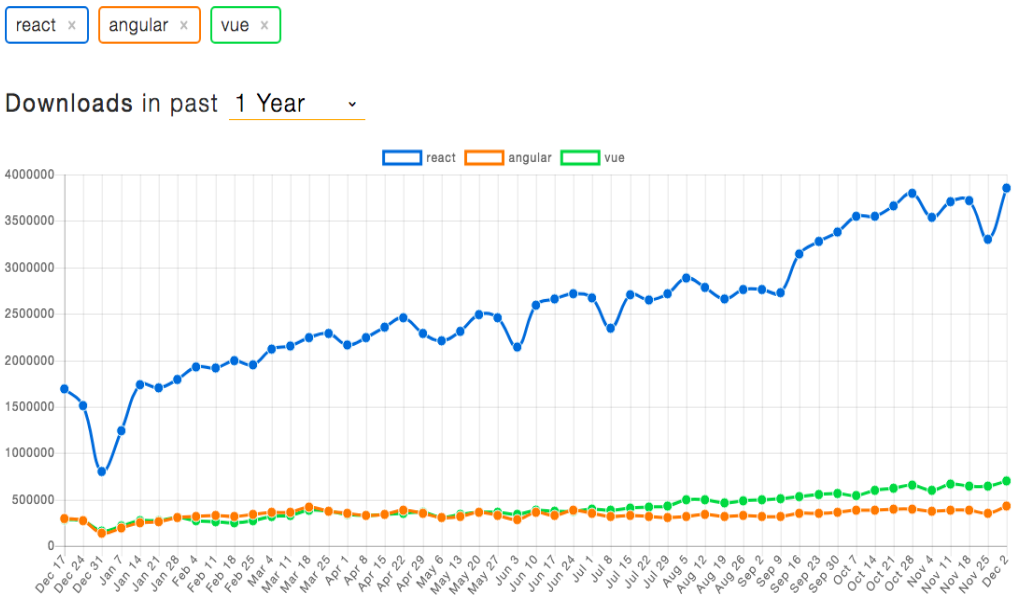
Naravno, svaki okvir ima i svoje nedostatke te će neki ovdje biti nabrojeni. Prvo se može smatrati nedostatkom, ali to ovisi o programeru, a to je jako brzi razvoj samog okvira kroz nove mogućnosti, promjene starih te kroz nove alate. To uzrokuje da se programer koji se koristi ovim okvirom konstantno mora informirati i učiti kako se radi s okvirom i novim alatima. Iz tog nedostatka proizlazi još jedan, a to je nedostatak dokumentacije. Zbog brzog razvoja pokatkad nema vremena za pravilnu i potpunu dokumentaciju koja bi potpuno opisala novi alat ili mogućnost. Sljedeći je nedostatak isto tako subjektivan, a to je miješanje HTML i JavaScript koda. React se koristi JSX ekstenzijom koja omogućuje to miješanje koje, naravno, ima svoju ulogu u okviru, ali neki programeri smatraju da je zbunjujuće i da nije lagano za učenje [13].

5.4. Vue.js

Vue.js je posljednji okvir koji će biti opisan i ima neke sličnosti sa React okvirom. On omogućuje vezanje podataka i ponovnu iskoristivost komponenti. Uz sve to, riječ je o vrlo maloj biblioteci koja je kompaktna i ne zahtijeva nikakve vanjske priključke ni ekstenzije, ali istodobno ima na izbor dodatne pomoćne biblioteke i alate. Okvir se primarno upotrebljava za izgradnju sučelja i vrlo je popularan u Kini. Sada će ukratko biti opisane neke prednosti okvira. Prva je u tome što je riječ o vrlo jednostavnom okviru koji se može relativno brzo svladati, odnosno ima laganu krivulju učenja. Osim toga, taj se okvir lako može integrirati u postojeći projekt zbog mogućnosti izolacije komponenata koje se mogu iskoristiti u više projekata. Činjenica da je sam okvir mali i da većina alata koji se rabe uz njega nisu veliki omogućuje brzo otpremanje na preglednik, što ujedno znači i veće brzina, a to ovom okviru daje prednost nad većim okvirima. Ovo je relativno novi okvir pa još nema mnogo toga čime bi se istaknuo nad konkurencijom. Okvir također ima aktivnu zajednicu koja se razvija, ali je još uvijek relativno mala [14].
Nedostatak ovog okvira jest u tome to što je još relativno nov i nema mnogo komponenata, odnosno treba još vremena da bi se razvila nova i bolja verzija tog okvira [14].

5.5. Usporedba

Sva tri prije opisana okvira rabe se često, ali je trenutačno najpopularniji React koji je naglo postao vrlo korišten. Iza njega je Angular i Vue.js. Na slici 8. prikazan je broj preuzimanja za svaki od gore nabrojanih okvira.



Slika 8. Graf količine preuzimanja programskih okvira (React, Angular, Vue.js) u godini 2018. (izvor:https://blog.bitsrc.io/11-vue-js-component-libraries-you-should-know-in-2018-3d35ad0ae37f,2018)

Graf prikazuje broj preuzimanja pojedinog okvira tijekom godine 2018. Može se vidjeti da React ima mnogo više preuzimanja od ostalih okvira, ali to nužno ne znači da je najkorišteniji, već samo da je imao najviše preuzimanja. Naime, AngularJS je stariji okvir koji već ima veliku bazu korisnika i sve teže privlači nove zbog toga što je krivulja učenja za taj okvir poprilično strma i samim time taj okvir postaje manje pristupačan. S druge strane, React isto tako nije najlakši za učenje, ali je lakši i brži i usto vrlo dobro podržava dinamički razvoj i brzu komunikaciju s korisnicima. Vue.js je još uvijek relativno nov okvir koji je u usponu i sakuplja sve veću zajednicu aktivnih korisnika. Odabir okvira ovisi o tome što treba programeru i koliko mu to brzo treba, na primjer ako programer koji nema iskustva ni s jednim od gore navedenih okvira želi naučiti neki od njih, treba dobro razmisliti o tome koji će odabrati. Ako mu treba jednostavan i lagan okvir koji se lako integrira i dobar je za izradu sučelja, može naučiti Vue.js, ako mu treba okvir koji podržava dinamičku izradu stranica, brzu reakciju na zahtjeve korisnika i kojeg dobro podržavaju mnogi alati i radi veću aplikaciju, može odabrati React i, naposljetku, ako ima mnogo vremena da nauči okvir koji ima mnogobrojne mogućnosti i vrlo opsežnu dokumentaciju, može naučiti AngularJS. Sve u svemu, sva ta tri okvira imaju svoju ulogu i moguću budućnost kao JavaScript okviri, ali se treba uzeti u obzir da se novi okviri pojavljuju brzo i možda će neki drugi okviri biti popularniji u budućnosti. U nastavku će biti uspoređena dva popularnija okvira od spomenutih triju.

5.5.1. Usporedba React i AngularJSa

Tablica ispod prikazuje usporedbu između React i AngularJS okvira.

Tablica 1. Usporedba AngularJS i React okvira (izvori: [15][28])

Usporedba okvira:	AngularJS	React
Autor (Tvrtka)	Google	Facebook
Tip okvira	Potpuni okvir	JavaScript biblioteka
Implementacija	Jednostavna	Zahtijeva dodatne biblioteke i alate
Broj dostupnih alata	Malen	Veliki
Jezik	JavaScript	JSX (JavaScript XML)
Krivulja učenja	Teška	Lagana
Učitavanje (eng. Rendering)	Na strani klijenta	Na strani servera
Struktura projekta	Stroga (MVC uzorak)	Nema, zahtijeva dodatne alate poput Flux alata (veća sloboda kod izbora strukture)
Vezanje podataka (eng. Data binding)	Obostrano vezanje (eng. Two way data binding)	Jednosmjerno vezanje prema dolje (eng. Downward data binding)
DOM	Obični DOM	Virtualni DOM (znatno brži)
Dokumentacija	Opsežna	Manje opsežna
Ubrizgivanje ovisnosti (eng. dependency injection)	Podržava ubrizgivanje ovisnosti	Potreban je dodatni alat

Koji okvir koristiti je teško odgovoriti, AngularJS je potpuniji i veći okvir s više mogućnosti, dok je React lakši (zauzima manje memorije), omogućava veću slobodu te je jednostavniji za naučiti i u nekim situacijama ima bolju performanse od AngularJS okvira (npr. kod stranica s puno prometa i koje zahtijevaju promjenu u stvarnom vremenu (eng. real-time)).[15] [28]

6. Web-aplikacija za vremensku prognozu

Ovo poglavlje jest vezano uz samu aplikaciju za vremensku prognozu i u ovom poglavlju će biti opisana sama aplikacija, njezina baza podataka i funkcionalnosti.

6.1. Opis aplikacije

U ovom će poglavlju ukratko biti opisana aplikacija. Tema je aplikacije vremenska prognoza na području Hrvatske, a riječ je o web-aplikaciji u .Net okviru, točnije, u ASP.NET okviru i u MVC tipu aplikacije (MVC uzorku). Sama je aplikacija podijeljena u tri glavne mogućnosti.

Prva je od tih triju najbitnija i to je sama vremenska prognoza za područje Hrvatske. Sam prikaz prognoze bit će na dva načina te će postojati i odabir između jednodnevne i peterodnevne prognoze. Prvi je oblik prikaza tablični i on koji uključuje sve podatke i detaljniji je od drugog prikaza. Drugi oblik uključuje graf, odnosno grafički prikaz prognoze. Graf prikazuje većinu toga što i tablica, uz napomenu da se na osi x nalazi vrijeme, a na osi y temperatura, te uz to, u slučaju da je predviđena padalina, postoje i mali stupići koji prikazuju padaline. Biti će prikazani podatci o samoj temperaturi, vlažnosti, vjetru i padalinama. Sami će podatci biti prikupljeni sa API-ja pod nazivom OpenWeather (hrv. Otvoreno vrijeme) koji nudi dohvat prognoze za pet dana bez naplate. Što se tiče dostupnosti prognoze, ona će pokrivati velik broj mjesta, ali, naravno, neka mjesta (većinom manja, odnosno s malim brojem stanovnika) nisu pokrivena jer se za njih ne može dohvatiti prognoza. Svi će podatci biti pohranjeni u bazu podataka te će se prikupljati svaka tri sata u određena doba dana (tako da bi podatci bili što je moguće ažurniji). Taj dohvat će se obavljati u zasebnoj dretvi neovisno o korisniku. Korisnik će na samoj stranici, naravno, morati odabrati mjesto tako da upiše naziv mjesta i morat će odabrati jednu od opcija za prikaz prognoze (jednodnevna, peterodnevna). Ako korisnik pokuša poslati nepotpun zahtjev ili u njegovu zahtjevu grešaka, bit će upozoren na greške i morati će ih ispraviti ako želi poslati zahtjev.

Druga je mogućnost karta na kojoj korisnik može vidjeti trenutačno vrijeme za neka mjesta u Hrvatskoj (broj mjesta koja će biti dostupna ovom mogućnošću jest manji jer je ograničen broj dohvata i ograničen je broj mjesta koja su prepoznata, odnosno za koje se šalje zahtjev API-ju). I u ovoj se mogućnosti primjenjuje OpenWeather API, ali se podatci ne spremaju u bazu, već se dohvaćaju kako se korisnik šeće po mapi. Naravno, mapa će imati mogućnosti micanja i zumiranja, ali će biti grubo ograničena na područje Hrvatske, kako korisnici ne bi dohvaćali

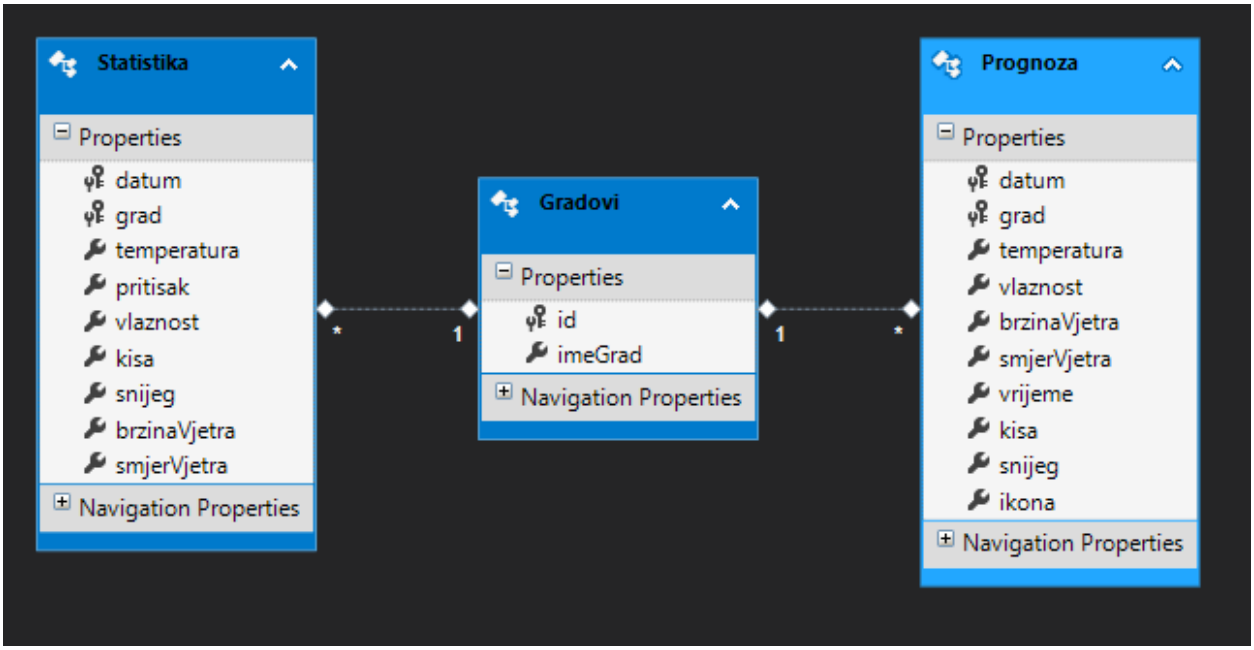
podatke za neke druge zemlje, te tako vrlo brzo potrošili maksimalan broj zahtjeva prema API-ju. Prikaz vremena bit će u obliku ikona na koje korisnik može kliknuti kako bi detaljnije vidio podatke. Podatci će biti unutar malog prozorčića koji će se otvoriti na klik. Sama će karta imati i opciju da se poveća na veličinu ekrana (eng. full screen), dosta tih mogućnosti uvijek je dostupno kada se rabe Google karta API-ji. Nažalost, zbog nekih promjena u vezi s tim API-jima sama će karta biti prekrivena oznakama da se može koristiti samo za razvoj i imat će sivi sloj preko sebe, ali i dalje će se moći normalno koristiti.

Treća i zadnja mogućnost jest statistika. Riječ je o jednostavnoj statistici koja uključuje tri moguće vrste statistike. Prva je maksimum koji prikazuje maksimalnu vrijednost za neki podatak u nekom razdoblju. Sljedeća je minimum koja prikazuje minimum i zadnja je prosjek koji prikazuje prosjek nekog podatka. Postojat će izbor između više podataka, a neki su od njih: temperatura, tlak, vlažnost, padaline itd. Naravno, već je prije bilo spomenuto da će podatci biti dohvaćeni za neko razdoblje, a ta će razdoblja biti: za godinu, za godinu po mjesecima (prikaz podataka za godinu, ali su podaci podijeljeni po mjesecima, npr. prosjek temperature za svaki mjesec u godini) i za mjesec. U ovoj će se mogućnosti koristiti drugi API pod nazivom Meteostat (meteorološka statistika) koji nudi statistiku za vremenske stanice. Koristit će se podatci s vremenskih stanica blizu ili unutar nekog mjesta (broj je tih mjesta ograničen, pa će i sam izbor mjesta za koje korisnik želi dohvatiti statistiku biti ograničen). Nažalost, sam API nije vrlo pouzdan i često podatci ne postoje, pa će dosta često rezultat pretrage biti nula, što će označivati da najvjerojatnije ne postoje podatci za traženu statistiku u bazi. Naravno, kao i kod prve mogućnosti postojat će obrazac koji će korisnik ispuniti i, ako nešto nije uredu s popunjenim poljima, biti će javljeno što nije u redu da bi se to moglo ispraviti i da bi korisnik mogao poslati zahtjev.

To su sve mogućnosti aplikacije kojima će korisnik imati pristup i s pomoću kojih će moći pristupiti željenim podatcima, je li riječ o prognozi za neko mjesto, pregledu trenutačne prognoze na karti ili o nekakvoj statistici.

6.2. Baza podataka

U ovom će poglavlju biti opisana sama baza podataka koja se rabi u sklopu aplikacije. Na slici 9. Prikazan je ERA model baze podataka. Baza podataka sama po sebi nije velika, sastoji se od samo triju tablica koje služe za dvije od triju mogućnosti aplikacije (za prognozu i statistiku).



Slika 9. ERA model baze za vremensku prognozu

Sam ERA model je relativno jednostavan i tablica koja ima najviše veza jest gradovi, pa će ona biti prva opisana. Sama je struktura tablice jednostavna i ima dva stupca (id i sam naziv grada, odnosno njegovo ime, uz napomenu da dosta zapisa u ovoj tablici nisu gradovi, nego i manja mjesta, tj. sela). Postoji nekoliko razloga zašto ova tablica nije jednostavno ukomponirana u druge dvije. Prvi je razlog u tome što se imena gradova ponavljaju često i općenito `int` tip podatka zauzima manje mjesta od naziva mjesta (nije uvijek tako), drugi je razlog u tome da se u toj tablici nalaze svi gradovi za koje aplikacija dohvaća prognozu, odnosno prognoza se neće moći dohvatiti ni za jedno mjesto koje nije u toj tablici, što osigurava da se ne mora slati zahtjev API-ju kako bi bilo utvrđeno postoji li uopće prognoza za dano mjesto, nego sigurno postoji prognoza za sva mjesta koja se nalaze u tablici.

Iduća tablica koja će biti opisana jest prognoza. U toj se tablici nalaze svi podatci koji se prikazuju korisniku kada zatraži prognozu za neko mjesto. Kao što se vidi na slici, ta tablica ima dvokomponentni primarni ključ. Prvi član ključa jest datum, a drugi je grad. Ta je kombinacija jedinstvena jer se prognoza dohvaća u intervalima od tri sata i nikad ne bi smjelo doći do toga da postoji zapis s istim vremenom za isto mjesto, odnosno ne mogu postojati dva zapisa za

Zagreb u tri sata popodne, pa takav ključ osigurava da neće doći do toga. Većina je vrijednosti u toj tablici brojčana, osim datuma koji je tipa datum-vrijeme (eng. TimeDate) i ikone i vremena. Vrijeme je stupac u koji se upisuje vremenska prilika u nekom mjestu i taj je opis, naravno, na hrvatskom (nasreću, korišteni API je podržavao hrvatski jezik).

Zadnja je tablica statistika i ona je vrlo slična tablici prognoza, ali ima neke razlike (tlak i snijeg koji ne označuje količinu snijega, nego dubinu, a kiša označuje sve padaline). Kada se pogledaju obje tablice, moglo bi se reći da je to mogla biti jedna tablica s dodatnim poljima koja bi za prognozu jednostavno bila nula ili ništa (eng. null). Najveći razlog zašto te tablice nisu spojene jest u tome da ne bi mogla postojati dva zapisa s istim mjestom i datumom (bez promjene primarnog ključa, možda dodatkom običnog ID-ja, ali onda bi mogli lakše nastati duplikati zapisa), sljedeći je razlog razlika u vremenskim intervalima u zapisima, odnosno za statistiku se dohvaćaju podatci u intervalima od jedan sat, a za prognozu tri sata, što može uzrokovati zbunjenost.

To su sve tablice, kako je na početku bilo rečeno, riječ je o relativno jednostavnom modelu, ali je dovoljno opsežan za potrebe aplikacije. U nastavku će biti opisani API-ji uporabljeni za dohvat podataka, te Hangfire dodatak i njegova baza podataka.

6.2.1.1. OpenWeather API

Ovaj se API u aplikaciji uporabljuje za dohvat prognoze na području Hrvatske. Sam API ima besplatnu opciju i razne plaćene opcije, koje će malo kasnije biti opisane. API je relativno jednostavan za uporabu i ima razne mogućnosti dohvata, od kojih će neke zanimljivije opcije biti nabrojene: trenutačno vrijeme, prognoza za četiri dana s vremenskim intervalom od jedan sat, dnevna prognoza za šesnaest dana (interval cijeli dan), peterodnevna prognoza s intervalom od tri sata, povijesni podatci i statistika. Naravno, mnoge od tih opcija nisu dostupne za besplatnu verziju. One koje su glavne za ovu aplikaciju dostupne su besplatno, a to su trenutačno vrijeme i prognoza za pet dana, ali je broj zahtjeva ograničena na šezdeset u minuti. Najskuplja plaćena opcija stoji dvije tisuće dolara, što je poprilično mnogo novca. Sami su podatci dostupni u tri formata, a to su JSON, XML i HTML. U aplikaciji se rabi JSON, odnosno ta je opcija odabrana za dohvat. Ostale bitne opcije jesu mjerne jedinice i podrška za više jezika (uključujući i hrvatski). Sami se podatci mogu tražiti preko imena mjesta, njegova poštanskog broja, po geografskim koordinatama ili ID-u (popis ID-ova postoji na stranici). Iako je često dovoljno specificirati samo ime mjesta, radi sigurnosti je bolje staviti i oznaku države. Sam API vrlo je dobro pokriven u smislu dokumentacije, na njihov stranici (opisi, primjeri, mogućnosti), ali i na samom internetu, odnosno drugim stranicama (razni primjeri koji se mogu izravno primijeniti u

aplikaciji, pa čak i primjeri rada API-ja s drugim API-jima). Taj API podržava dosta drugih API-ja, uključujući nekoliko API-ja za karte. Početak korištenja API-jem svodi se na kreiranje računa na stranici i, naravno, potvrde računa nakon koje se dobije API ključ koji se može rabiti (može se tražiti promjena ključa, što može biti korisno ako netko krade već ograničene zahtjeve preko korištenog ključa). Ovaj je API također vrlo pouzdan, vrlo rijetko nema tražene podatke i malokad je usluga nedostupna [25].

Sam URL API stranice jest https://openweathermap.org/api, a u nastavku će biti prikazan primjer zahtjeva i dio odgovora na taj zahtjev (opcija peterodnevna prognoza).

```
api.openweathermap.org/data/2.5/forecast?q=Zagreb,hr&units=metric&mode=json&lang=hr&appid=af8fb6bdc62c7991812b3da96e86fc5d
```

Sam zahtjev ima nekoliko parametara. Prvi jest sam naziv mjesta i oznaka države, nakon toga je parametar za sisteme mjerenja (u ovom slučaju metrički), nakon čega je parametar za format u kojem će odgovor biti. Zadnja dva parametra su za jezik i sam API ključ. Ispod je prikazan dio odgovora na zahtjev.


```
{
    "cod": "200",
    "message": 0.0075,
    "cnt": 40,
    "list": [
        {
            "dt": 1568116800,
            "main": {
                "temp": 22.09,
                "temp_min": 21.84,
                "temp_max": 22.09,
                "pressure": 1018.06,
                "sea_level": 1018.06,
                "grnd_level": 981.79,
                "humidity": 47,
                "temp_kf": 0.25
            },
            "weather": [
                {
                    "id": 803,
                    "main": "Clouds",
                    "description": "isprekidani oblaci",
                    "icon": "04d"
                }
            ],
            "clouds": {
                "all": 80
            },
            "wind": {
```

```
                "speed": 1.52,
                "deg": 119.018
            },
            "sys": {
                "pod": "d"
            },
            "dt_txt": "2019-09-10 12:00:00"
        }]
"city": {
        "id": 3186886,
        "name": "Zagreb",
        "coord": {
            "lat": 45.8132,
            "lon": 15.977
        },
        "country": "HR",
        "population": 784900,
        "timezone": 7200,
        "sunrise": 1568089661,
        "sunset": 1568135948
    }
}
```


Odgovor počinje sa HTTP statusnim kodom nakon čega slijedi lista sa vremenskim podacima (sadrži vremenske podatke i datum na kojeg se podaci odnose, lista prikazana iznad ima samo jednog člana jer svi članovi imaju istu strukturu, pa nema potrebe pokazivati više od jednog člana). Odgovor završava s podacima o gradu za kojeg je tražena prognoza.

6.2.1.2. Metostat API

Metoestat je stranica s raznim klimatskim podatcima i nudi API za povijesne podatke. Podatci su dostupni za vremenske stanice koje je potrebno pronaći za pojedina mjesta, neka mjesta, naravno, imaju i više stanica (pogotovo ako je riječ o velikim gradovima u svijetu). Nakon što je lociran ID stanice dostupne su razne opcije dohvata, od dohvata za svaki sat, pa do mjesec dana (u aplikaciji se primjenjuje dohvat za svaki sat). Ovaj je API potpuno besplatan za bilo koga i dostupno je dvjesto zahtjeva u minuti. To što je besplatan nije nužno i najbolja stvar, jer usluga nije garantirana i podatci nisu dostupni za sve mjesece i godine, te je i sama količina mjesta koja taj API podržava ograničena jer moraju imati vremensku stanicu u blizini, te ne postoji brz način da se nađu sve stanice u državi, nego se mora ići mjesto po mjesto ili tražiti podatke o vremenskim stranicama na drugome mjestu. Na stranici je detaljno opisan svaki oblik dohvata sa svim mogućnostima te postoje gotovi primjeri zahtjeva. Od opcija bitne su vremenska zona i format zapisa, koji je u slučaju aplikacije JSON. Zahtjev za ključem isto je tako vrlo jednostavan i ključ se može odmah nakon registracije upotrebljavati. Ovaj API nije ni

približno pouzdan kao OpenWeather API i zato je njegovo uključivanje u aplikaciju bilo malo teže, ali ovaj dohvat ima vrlo zanimljivu mogućnost, a to je dohvat goleme količine podataka s jednim zahtjevom, ta mogućnost nije iskorištena u aplikaciji zbog komplikacija koje dolaze s tom mogućnošću (pokatkad ne dohvati sve podatke, nego vraća parcijalni JSON, što može uzrokovati greške, nije lagano pratiti za što su dohvaćeni podatci, a za što nisu itd.) [26].

6.2.1.3. Hangfire dodatak

Hangfire je dodatak za C# aplikacije koji dopušta kreiranje poslova (ti se poslovi pohranjuju u bazu podataka te se pravodobno izvode u obliku dretava). Postoje četiri vrste poslova i u nastavku će biti nabrojene i kratko opisane. Prva je ispali i zaboravi posao (eng. fire and forget) koji se, kako i ime sugerira, izvršava što je prije moguće i nakon toga se briše. Druga je vrsta odgođeni posao (eng. delayed job) koji se izvršava nakon što prođe željeno vrijeme. Treća su vrsta ponavljajući poslovi (eng. Recurring jobs) koji se ponavljaju u određenim intervalima ili u određena doba dana (takvi poslovi se primjenjuju u aplikaciji). Četvrti i zadnji posao jest posao nastavljanja (eng. continuations) koji se izvršava nakon što je završen posao roditelj. Usto, postoje još neki poslovi, ali ti su za plaćenu verziju, pa neće biti nabrojeni. Ovaj dodatak znatno skraćuje vrijeme potrebno za ostvarivanje rada s dretvama i ima super mogućosti, ali ima i svoje nedostatke. Sama dokumentacija za taj dodatak je poprilično slaba i nedovoljno objašnjena (npr. piše što neka naredba radi, ali ne definira se točno što to sve pokriva), problem je čak i na drugim stranicama naći rješenja, a sam forum stranice često se svodi na „greška se događa zbog programa treće strane i ne mogu vam pomoći“. Ovaj dodatak svoje poslove sprema u bazu u razne tablice (nije pretjerano intuitivno, već se svodi na tehniku pokušaj i vidi) te se preko njih do neke razine mogu otkrivati greške u radu dodatka. Svaki posao ima nekoliko stanja (procesiranje, uspješno, zakazivanje, prekinut itd.) i vrlo je bitno pratiti kako se ona mijenjaju da se ne bi pojavile pogreške. Svi aktivni i završeni poslovi spremljeni su u tablicu posao (eng. job) i u toj se tablici vidi njihovo trenutačno stanje, te kada su počeli i završili, odnosno u toj se tablici se može pratiti izvođenje poslova. Sljedeća bitna tablica jest red za poslove (eng. JobQueue), u kojoj se nalaze svi poslovi koji čekaju svoje izvođenje, a jednom kad su pokrenuti, miču se iz reda, ali se tamo vraćaju ako je izvršavanje prekinuto ili neuspješno. Sljedeća bitna tablica jest set u kojoj se nalaze svi ponavljajući poslovi i njihovi nazivi (nazivi predstavljaju ID s pomoću kojeg se u aplikaciji pristupa tim poslovima). I zadnja tablica koja će biti opisana jest tablica stanja (eng. state) u kojoj piše u kojem je trenu neki posao bio u nekom stanju. Sami vremenski zapisi u toj bazi nisu korektni, u smislu da postoji vremenski interval između stvarnoga vremena izvođenja i onog u bazi. Samo uključivanje

dodatka obavlja se s pomoću NuGet upravitelja paketima, nakon čega se mora konfigurirati i kreirati server, a naposljetku se kreiraju sami poslovi [27].

6.3. Funkcinalnosti

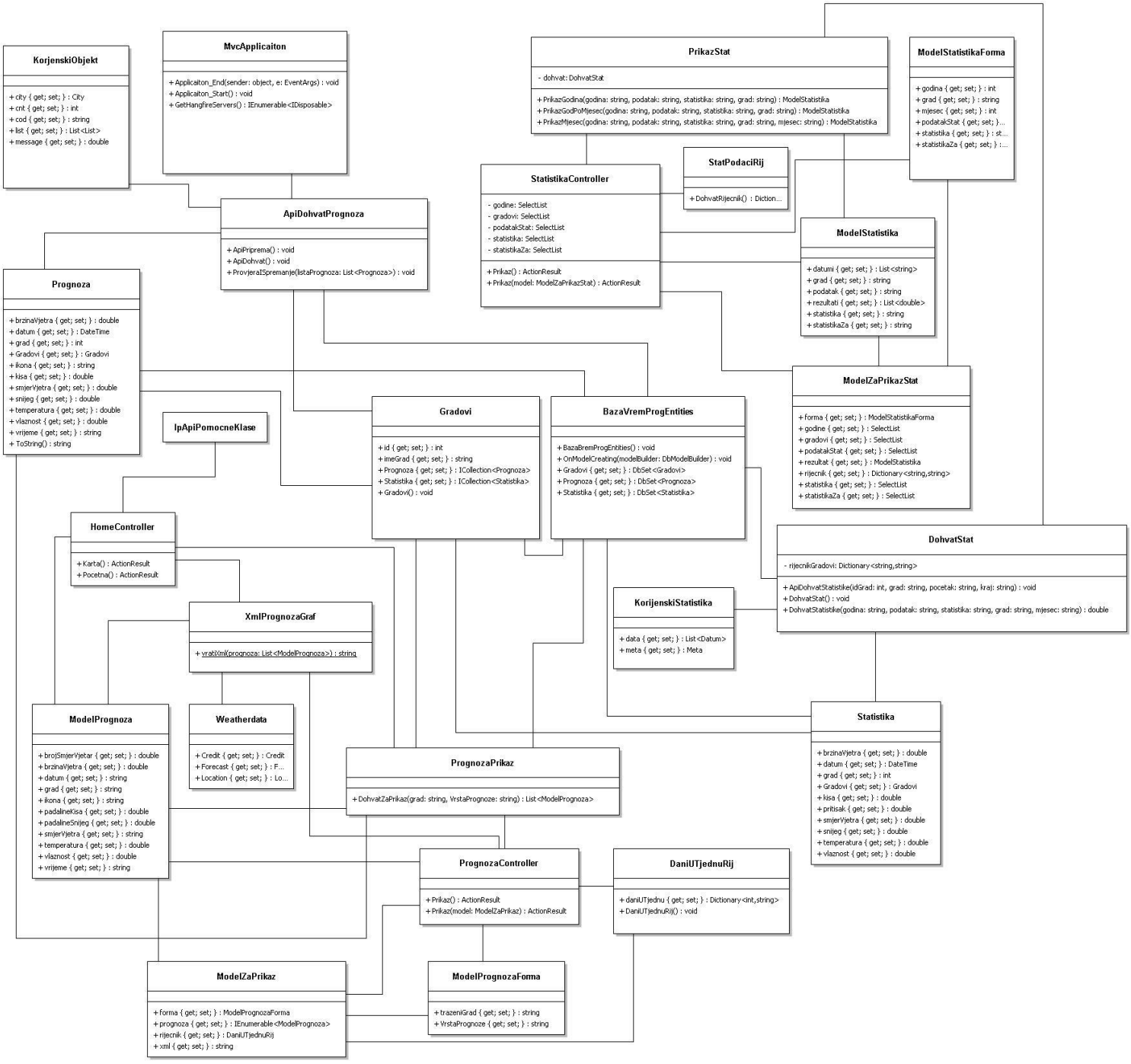
Na sljedećoj slici (slika 10.) prikazan je dijagram klasa za aplikaciju koji će biti opisan u nastavku. Na slici dijagrama su prikazane četiri glavne skupine klasa, a to su klase vezane za dohvat prognoze, klase vezane za prikaz prognoze, klase vezane za kartu i klase vezane za statistiku (naravno da se neke klase rabe i za dohvat i za prikaz prognoze, ali ovo je samo neka općenita podjela preko koje će biti opisane klase). Većina će klasa biti opisane općenito te će biti spomenute njihove metode (čemu služe i eventualno okvirno kojim se klasama koriste). Prva grupa klasa koje će biti opisane vezane su za dohvat prognoze. Prva je `MvcApplicaiton` (MVC aplikacija, ta je klasa zapravo generirana iz `Global.asax` datoteke i zato ima takav naziv). Ta klasa ima tri metode, a prva je `Applicaiton_end()` koja se poziva pri gašenju aplikacije (ta je metoda prazna i trenutačno ničemu ne služi). Druga je metoda `ApplicationStart()` koja se pokreće na početku aplikacije. U njoj se registriraju sve klase iz App_start datoteke (usmjerivanje, filtri i paketi) te se kreira server za Hangfire dodatak i poziva prvi posao (dretva, prvi dohvat podataka). Zadnja je metoda `GetHangfireServers()` koja služi za konfiguriranje servera (što uključuje i spajanje na bazu) te za uklanjanje postojećih ponavljajućih poslova (to nije tako u dokumentaciji Hangfire dodatka, već je bilo naknadno dodano, a razlog će biti objašnjen pri opisu mogućnosti i koda). Kako je bilo spomenuto, ta klasa pokreće prvi dohvat pa će sljedeća biti opisana klasa za dohvat koja se zove `ApiDohvatPrognoza`. Ta klasa ima tri metode, a prva je `ApiPriprema()`, koja prvo pokreće dohvat (drugu metodu po redu), a, kada je dohvat završen, pokreće ponavljajuće poslove (razlog je u tome da ne bi dva posla radila istodobno, pa zato prvo dohvaća podatke). Sljedeća je metoda `ApiDohvat()` koju pokreću poslovi i služi za dohvat podataka s API-ja te njihovo pretvaranje u oblik koji se može spremiti u bazu (kako bi se preveo JSON odgovor u zapis s kojim se može raditi, uporabljuje se klasa `KorijenskiObjekt`, koja je model za JSON i sadržava druge pomoćne klase koje nisu prikazane na grafu jer bi bio prevelik). Nakon pretvorbe pokreće se metoda `ProvjeraISpremanje()` koja prima listu tipa `Prognoza` i korektno sprema podatke u bazu. Provjera se odnosi na to da se provjerava je li došlo do promjene podatka s obzirom na posljednji dohvat, a, ako jest, zapisi u bazi se ažuriraju (i, naravno, dodaju se članovi koji uopće ne postoje u bazi). Ova se klasa koristi dvjema klasama koje je generirao Entity okvir, a to su `Prognoza` i `Gradovi`. `Gradovi` služe kako bi se mogli slati zahtjevi API-ju, a `Prognoza` je model

istoimene tablice u bazi (isto to su klase `Gradovi` i `Statistika`). Sve klase koje dohvaćaju iz baze podataka koriste se klasom `BazaVremProgEntities` koja služi za komunikaciju s bazom i generira je Entity okvira.

To pokriva sve klase vezane za dohvat, sljedeće na redu je prikaz. Prvo će biti opisana središnja klasa `PrognozaController` (dolje na dijagramu i, kao što samo ime govori, riječ je o kontroler klasi). Ta je klasa odgovorna za rukovanje zahtjeva veznih za prognozu. Ima dvije metode, od kojih je prva `Prikaz()` koja se pokreće kada korisnik odabere prognozu u izborniku preglednika. Ta metoda popunjava model (klasa `ModelZaPrikaz`) s dostupnim podatcima (u ovom slučaju to su samo instance klase `ModelPrognozaForma` i `DaniUTjednuRij`). Klasa `ModelZaPrikaz` sadržava instance klasa `ModelPrognozaForma`, `DaniUtjednuRij`, listu instanci `ModelPrognoza` i tekstualno svojstvo za XML. `ModelPrognoza` model je za prikaz podataka iz baze podataka (sama prognoza), `ModelPrognozaForma` jest model obrasca koji je prikazan korisniku, a `DaniUtjednuRij` sadržava rječnik s danima u tjednu. Sljedeća metoda kontrolera isto se tako zove `Prikaz()` i prima za parametar model obrasca (korisnikov zahtjev). Ta se metoda poziva kada korisnik napravi zahtjev (uspješno pošalje obrazac). Metoda također popunjava model, uz napomenu da, za razliku od prve, popunjava cijeli model, a, da bi to mogla, poziva metodu klase `PrognozaPrikaz` (za dohvat prognoze) i metodu klase `XmlPrognozaGraf`. Obje metode `Prikaz()` na kraju vraćaju pogled s modelom koji zatim korisnik vidi. Prvo ćemo opisati metodu klase `PrognozaPrikaz`, a ta metoda jest `DohvatZaPrikaz()` koja prima ime grada i tip prognoze (jednodnevna, peterodnevna). Ta metoda iz baze podataka dohvaća prognozu i prebacuje ju u model za prikaz (`ModelPrognoza`) te vraća listu instanci te klase. Sljedeća klasa jest `XmlPrognozaGraf` koja ima jednu metodu, a ta je metoda `vratiXml()` koja prima listu tipa `ModelPrognoza` i vraća tekst koji je XML oblika te se poslije uporabljuje da bi se kreirao graf. Ova se klasa koristi pomoćnom klasom `Weatherdata` (koja sadržava druge pomoćne klase) s pomoću koje se kreira XML tekst.

Time je završen prikaz prognoze i sljedeće će biti opisana klasa `HomeController` (kontroler koji je kreiran s projektom). Ta klasa ima dvije metode, a prva je `Karta()` koja se poziva kada korisnik odabere karta na stranici aplikacije. Sama metoda samo poziva pogled za kartu. Iduća je metoda `Pocetna()` koja se koristi pomoćnom klasom `IpApiPomocneKlase` kako bi prevela JSON i dobila lokaciju korisnika preko IP adrese te zatim dohvatila vremenske podatke i XML za tu lokaciju i iscrtala graf. Nažalost, to ne radi jer se stalno dohvaća lokalna IP adresa za koju se ne može naći lokacija.

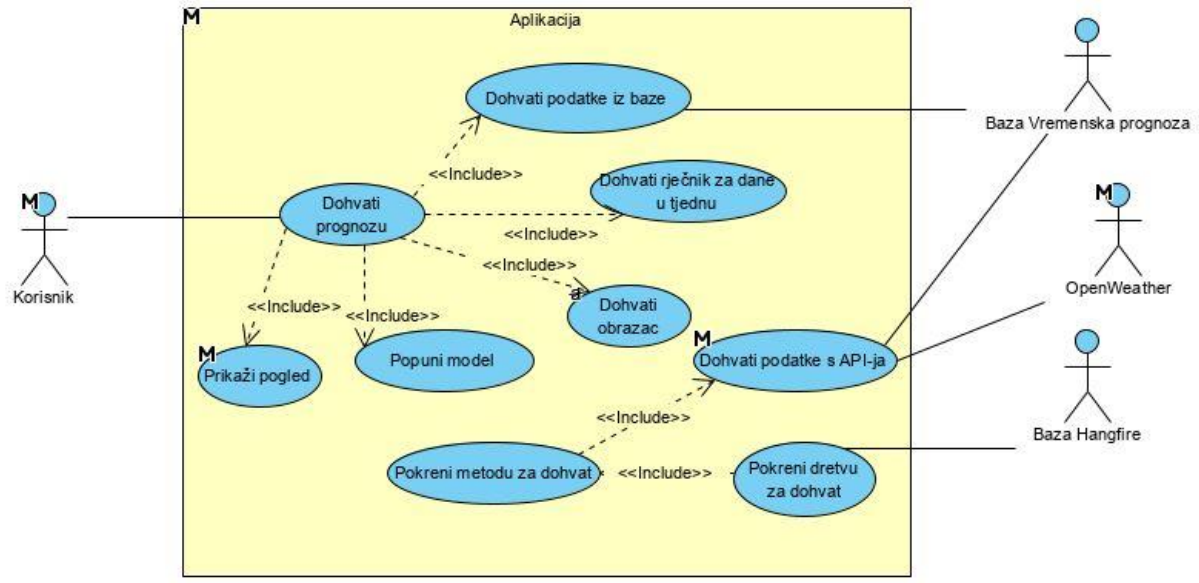
Zadnja cjelina jest statistika i opis će početi od klase `StatistikaController` (gore desno na grafu). Ta je klasa vrlo slična kontroleru za prognozu, koristi se svim klasama za model (obrazac, podatci i prikaz) i poziva dohvat podataka. Ima dvije metode s istim imenom `Prikaz()`, od kojih jedna služi za prikaz obrasca, dok korisnik prvi put dođe na statistiku na stranici, a druga za dohvat i prikaz rezultata (naravno i dalje prikazuje obrazac). Usto, ta klasa popunjava liste s izborima koje korisnik odabire na obrascu. Sljedeća je klasa `PrikazStat` čije metode primaju sve podatke s obrasca i vraćaju instancu klase `ModelStatistika`. Ova klasa Ima tri metode, svaku za jednu vrstu prikaza (npr. `PrikazMjesec()`). Sve te metode pozivaju metodu za dohvat klase `DohvatStat`. Ta klasa ima tri metode, jedna je konstruktor koji popunjava rječnik gradovi-vremenske stanice, druga je `DohvatStatistike()` koja dohvaća traženu statistiku iz baze podataka, a, ako nema podataka u bazi, poziva treću metodu `ApiDohvatStatistike()` koja dohvaća statistiku s API-ja i sprema je u bazu. Nakon što je završila metoda za dohvat s API-ja metoda `DohvatStatistike()` dohvaća i vraća traženu statistiku (zbog nekonzistentnosti API-ja to i dalje može biti nula). Time su ukratko opisane sve klase aplikacije.



Slika 10. Dijagram klasa aplikacije

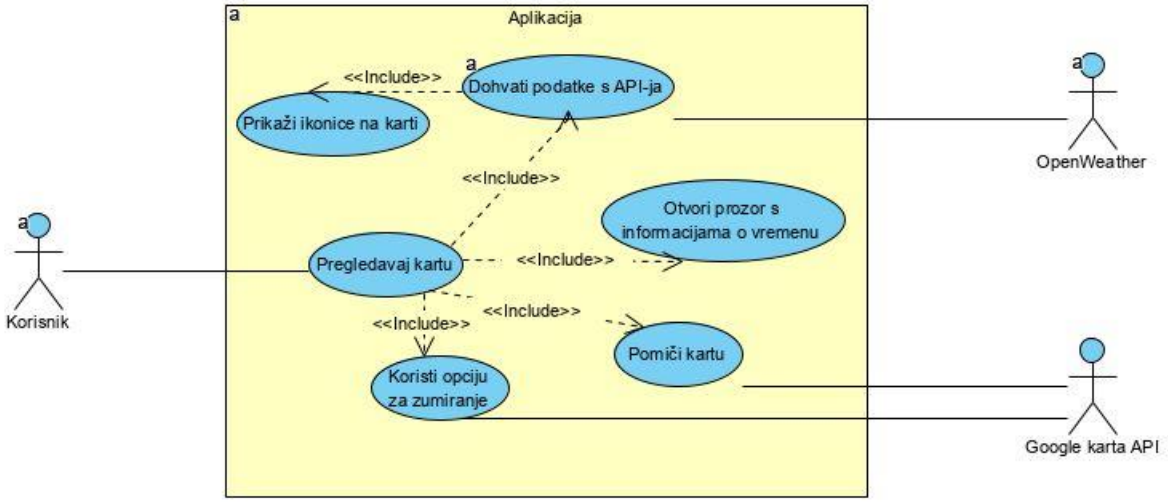
6.3.1. Dijagrami slučajeva uporabe

Dijagrami slučajeva uporabe općenito opisuju korisnikov pogled na aplikaciju i što je sve uključeno da bi taj pogled bio moguć. U nastavku teksta navode se tri dijagrama slučajeva uporabe, po jedan za svaku mogućnost.



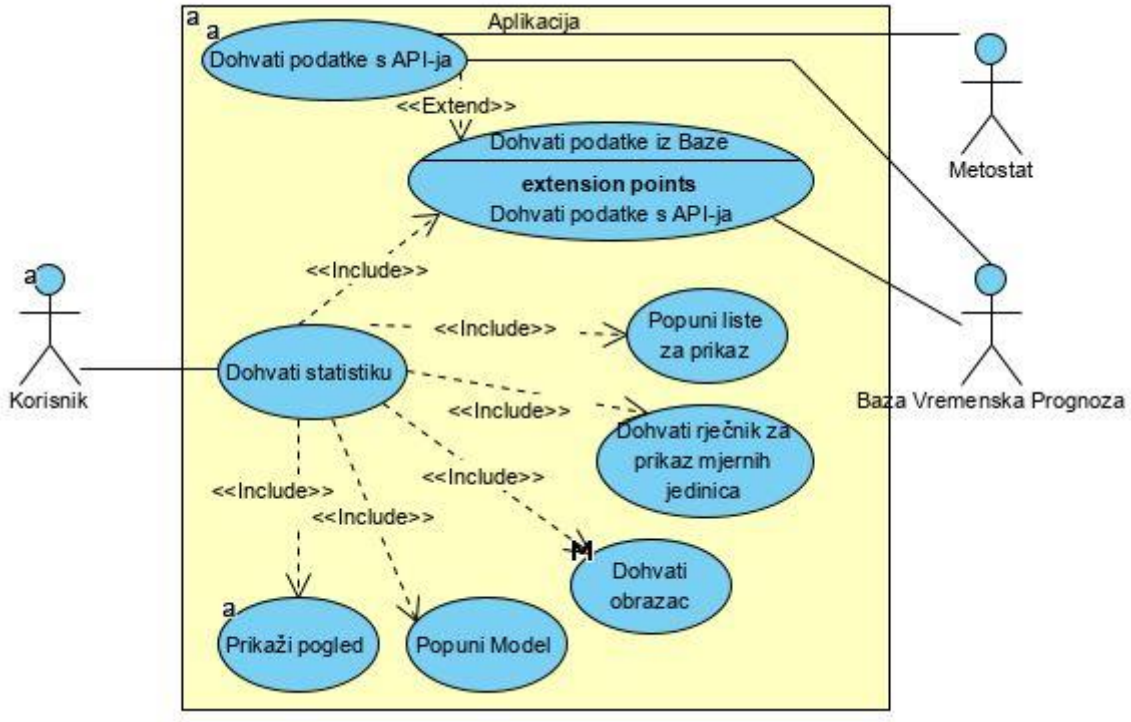
Slika 11. Dijagram slučajeva uporabe za mogućnost prognoze

Sada će biti ukratko opisani svi slučajevi unutar prikazanog dijagrama. S lijeve se strane nalazi korisnik koji pokušava dohvatiti prognozu, što je ujedno i prvi slučaj u samom sustavu aplikacija. Taj sustav uključuje mnogo slučajeva, a oni će biti opisani u nastavku. Uključuje dohvat podataka iz baze podataka te za to postoji glumac (eng. actor) baza podataka vremenska prognoza na desnoj strani. Uključuje dohvat rječnika za dane u tjednu, dohvat obrasca i također uključuje popunjavanje modela za prikaz i sam prikaz modela. Sve je to vezano za gornji dio grafa, a donji se dio odnosi na dohvat podataka koji inicijalizira glumac pod nazivom Baza Hangfire. Iz baze podataka se uzimaju poslovi i pokreću se odgovarajuće dretve. Te dretve izvode metoda za dohvat i zato je ta metoda uključena u pokretanje dretve, a sama metoda uključuje dohvat podataka s API-ja OpenWeather koji je prikazan kao glumac i, naravno, podatci se spremaju u bazu. Naravno, ovaj i ostali dijagram poprilično su općenit opis rada aplikacije koji će biti više objašnjen dijagramima aktivnosti i samim opisom mogućnosti i koda.



Slika 12. Dijagram slučajeva korištenja za mogućnost karte

Gore prikazani dijagram je za mogućnost pregleda karte. Korisnik pregledava kartu te to uključuje dohvat podataka s OpenWeather API-ja. Korisnik može micati kartu te zumirati i od zumirati, što je moguće zbog Googleove API-ja za karte, a također može otvoriti prozor s informacijama o vremenu klikom na ikonice koje se generiraju nakon samog dohvata.



Slika 13. Dijagram slučajeva korištenja za mogućnost statistike

Treći i zadnji dijagram slučajeva korištenja jest za statistiku i taj je dijagram vrlo sličan prvom. Korisnik pokušava dohvatiti statistiku, a to uključuje dohvat podataka iz baze. Ako ne postoje podatci u bazi, oni se dohvaćaju s API-ja Meteostat, te se, naravno, spremaju u bazu. Usto, potrebno je popuniti liste za prikaz (godine, gradovi itd.), te dohvatiti rječnik s nazivima podataka i mjernim jedinicama. Dohvati statistiku također uključuje dohvat obrasca,

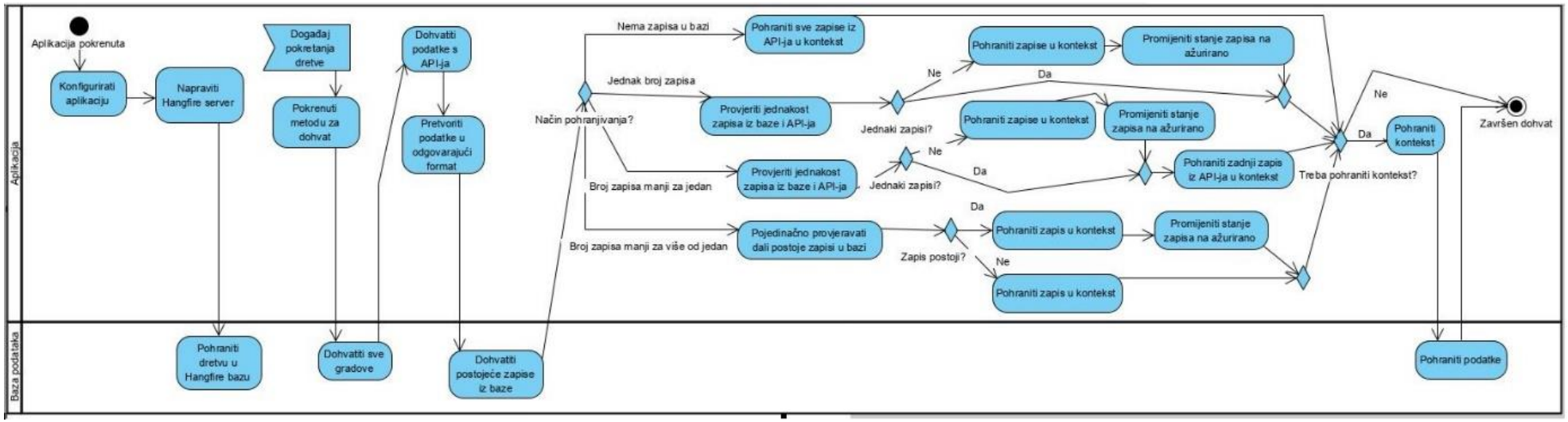
popunjavanje modela i prikaz pogleda. Sve u svemu, dohvat statistike dosta je sličan dohvatu prognoze, s jednom razlikom, a to je da se za dohvat prognoze primjenjuju dretve za dohvat s API-ja, a kod statistike to se radi ako podatci ne postoje u bazi.

6.3.2. Dijagrami aktivnosti

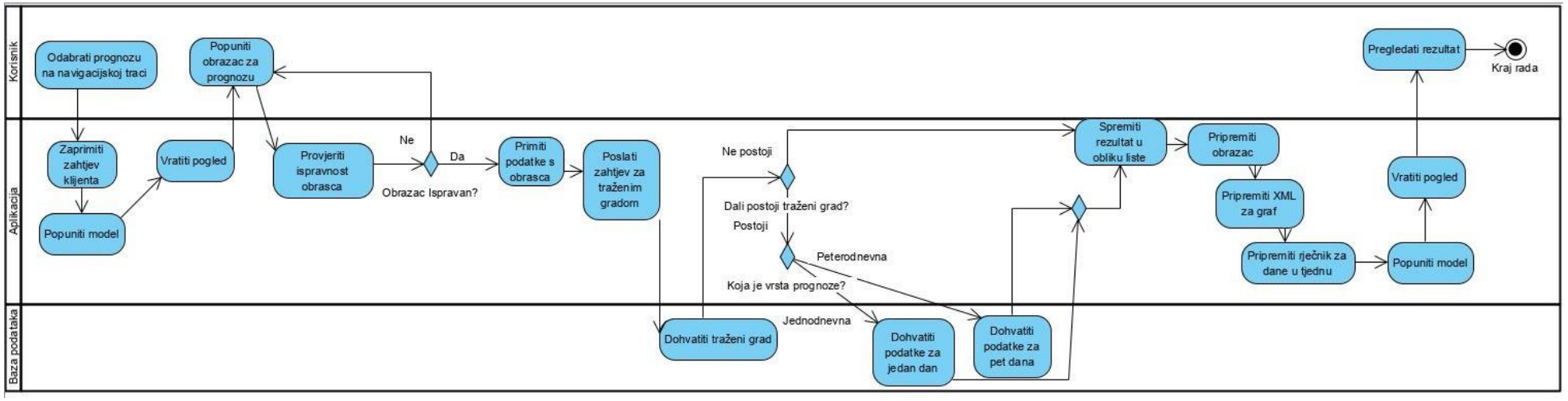
Dijagramima aktivnosti će biti detaljnije opisane mogućnosti iz dijagrama slučajeva. U nastavku će biti prikazani i opisani dijagrami aktivnosti. Prvo će biti opisan dijagram prikazan na slici 14. Sve počinje pokretanjem aplikacije, pri čemu se pokreće Global.asax klasa u kojoj se konfigurira aplikacija (registriranje područja, filtri, usmjerivanje,…) te se kreira Hangfire server i u red ubacuje prva dretva. Ta je, prva dretva drukčija od drugih jer pokreće sve ponavljajuće poslove (nije prikazano na dijagramu). Sve dalje na dijagramu događa se nakon što se pokrene jedna od dretvi (vremenski događaj). Sve te dretve pokreću istu metodu za dohvat u kojoj se prvo dohvaća lista svih gradova te se započinje petlja kako bi se dohvatili podatci (petlje se ne vide na grafu). Prvo se dohvaćaju podatci s API-ja koji se zatim iz pomoćnih klasa vade u klase koje je kreirao okvir Entity. Za svaki se grad dohvaćaju već postojeći podatci iz baze podataka (za prognozu) te se na temelju razlike broja zapisa između tih podataka i onih iz API-ja izvršavaju različiti slučajevi. Ako u bazi nema nijednog zapisa, tada se svi iz API-ja dodaju u kontekst. Ako je broj zapisa jednak, tada se provjerava jednakost zapisa (svih odjednom), a, ako su različiti, tada se u kontekst pohranjuju zapisi iz API-ja i njihovo se stanje mijenja na ažurirano (direktni prijevod stanja bi bio promijenjeno, što uzrokuje ažuriranje u bazi pri spremanju konteksta, ali je radi jednostavnosti napisano ažurirano). Ako je broj zapisa manji za jedan, tada se uspoređuju zapisi (odjednom, svi osim zadnjeg) i, ako postoji razlika, svi se zapisi spremaju u kontekst i mijenja im se stanje, pa se zadnji zapis sprema u kontekst (neovisno o jednakosti). Ako je razlika u zapisima veća od jedan, tada se ide kroz svaki zapis i provjerava se postoji li on u bazi. Ako dakle postoji, onda se pohranjuje u kontekst i mijenja mu se stanje, a inače se samo sprema u kontekst. Ako je u slučaju došlo do dodavanja u kontekst, onda se on sprema i podatci se pohranjuju u bazu, a, ako nije, tada se odmah završava dohvat za taj grad i kreće se s sljedećim (naravno, na grafu je prikazan kraj, odnosno završni čvor).

Idući dijagram je za prikaz podataka za prognozu (slika 15.). Dijagram počinje korisnikovim odabirom te se za njega učitava stranica, zaprimljen je zahtjev, popunjava se model (isto kao i kasnije, gdje je detaljnije prikazano što sve ide u model), te se vraća pogled. Korisnik potom ispunjava obrazac i šalje ga, nakon čega se provjerava njegova ispravnost i, ako je on nepravilno popunjen, tada se vraća korisniku i on mora popraviti obrazac. Kada je poslan pravilan obrazac, tada se na temelju primljenih podataka pokreće postupak dohvata podataka za prikaz. Prvo se pokuša dohvatiti traženi grad te, ovisno o tome postoji li on u bazi, izvršavaju se različiti slučajevi. Ako grad postoji, tada se gleda koji je tip prognoze

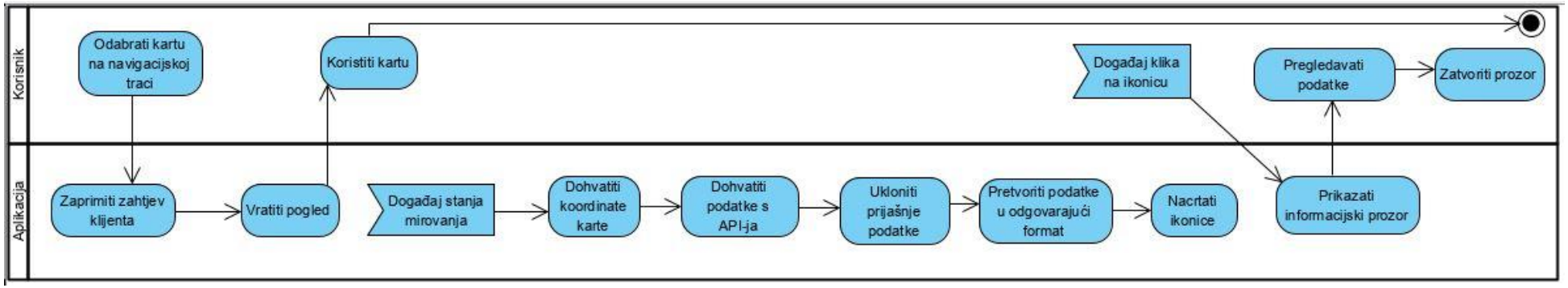
korisnik zatražio i ovisno o tome dohvaćaju se podatci iz baze podataka i spremaju u listu. Ako grad nije postojao, onda će lista biti prazna. Ti rezultati i ostali članovi modela za prikaz (obrazac, XML, rječnik) stavljaju se u model i pogled se prikazuje korisniku koji pregledava rezultat i nakon toga je završen rad.



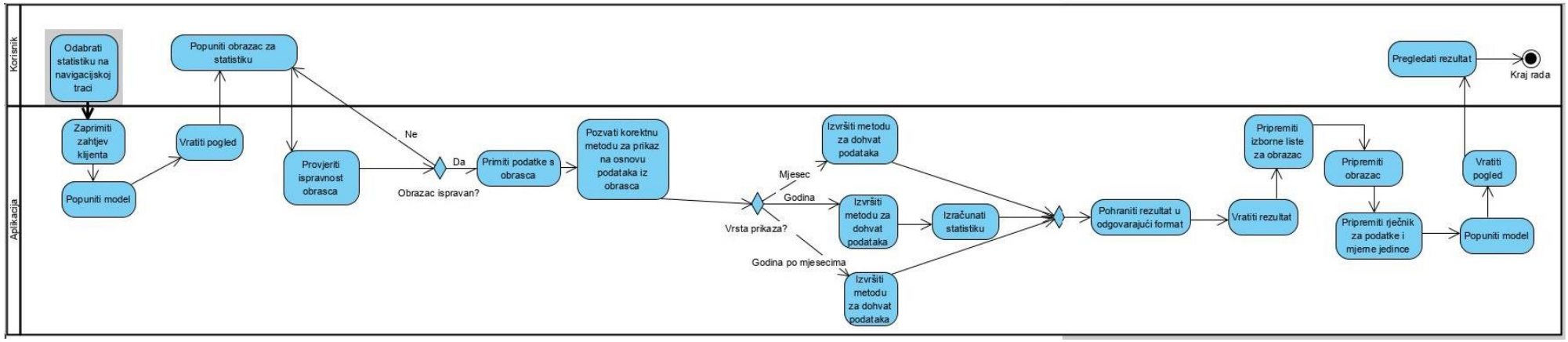
Slika 14. Dijagram aktivnosti za dohvat prognoze



Slika 15. Dijagram aktivnosti za prikaz prognoze



Slika 16. Dijagram aktivnosti za mogućnost karte



Slika 17. Dijagram aktivnosti za mogućnost statistike

Sljedeći dijagram je za mogućnost karte (slika 16.). Dijagram počinje korisnikovim odabirom te se zaprima njegov zahtjev i vraća odgovarajući pogled. Korisnik se koristi kartom i, kada završi s tim, rad se završava. Sve što se događa u ovoj mogućnosti jest reakcija na događaje. Prvi je događaj stanje mirovanja, odnosno kada korisnik prestane pomicati kartu (to se prvi put događa odmah kada se učita karta) ili kada zumira ili od zumira. Prvo što se događa nakon toga jest to da se dohvaćaju koordinate karte (rubovi karte koji su trenutačno vidljivi) te se na temelju tih koordinata dohvaća s API-ja. Kada je stigao odgovor, prvo se uklanjaju postojeći podatci (oni koji su već na karti) i zatim se ti novi podatci spremaju u odgovarajućem formatu i na kartu se dodaju ikonice (uključuje spremanje podataka u podatke same karte). Idući događaj jest kada korisnik klikne na jednu od ikonica. Tada se otvara prozor s informacijama, koje su bile spremljene pri događaju mirovanja. Korisnik pregledava informacije i, kada završi, zatvara prozor.

Zadnji dijagram prikazuje mogućnosti statistike (slika 17.). Dijagram počinje odabirom klijenta, zaprima se zahtjev i popunjava model i vraća pogled. Korisnik ispunjava obrazac i, ako on nije pravilan kada ga korisnik potvrdi, tada ga mora ponovno ispuniti. Ako je ispravan, primaju se podatci s obrasca i poziva se korektna metoda za prikaz, ovisno o tome koju je vrstu prikaza korisnik odabrao. Ako je odabrao prikaz za mjesec, tada se izvršava metoda za dohvat podatka te se poslije rezultat sprema u odgovarajući format i vraća. Ako je korisnik odabrao godinu, isto tako se dohvaćaju podatci (više puta, jedanput za svaki mjesec) te se na kraju provodi izračun, nakon čega je sve isto kao i u prijašnjem slučaju. Naposljetku, ako korisnik odabere prikaz godine po mjesecima, dohvaćaju se podatci i dalje je sve isto (razlika između ovog i prvog slučaja jest u tome da se u ovom slučaju dohvaćaju podatci za sve mjesece). Iako na grafu metoda za dohvat nije detaljno prikazana, svejedno će biti ukratko opisana. Prvo se iz baze podataka dohvaća ID grada i zatim se provjerava postoje li podatci za traženu statistiku. Ako ne postoje, tada se poziva metoda koja iz API-ja dohvaća podatke, stavlja ih u korektan tip podataka i sprema u bazu. Nakon tog dohvata ili ako podatci već postoje, dohvaća se tražena statistika (minimum, maksimum, prosjek) za traženi podatak (npr. temperatura) i vraća se rezultat (ova metoda uvijek dohvaća podatke za jedan mjesec). Povratkom na dijagram, sljedeće nakon vraćanja rezultata jest priprema izbornih lista za obrazac (liste koje će biti prikazane na obrascu te iz kojih će korisnik birati ono što želi), zatim se priprema obrazac, te rječnik i naposljetku se popunjava model i vraća pogled. Korisnik pregledava rezultat i nakon toga je kraj rada.

6.3.3. Glavne mogućnosti

U nastavku će biti opisane glavne mogućnosti uz prikaz određenih bitnijih dijelova koda.
Prva mogućnost jest prognoza i opis će započeti dohvatom prognoze, odnosno `Global.asax` klasom. U njezinoj metodi za početak aplikacije, konfigurira se aplikacija i pokreće Hangfire server i prva dretva. U nastavku će biti prikazane linije koda za kreiranje i brisanje poslova, te uporabu servera.

```
HangfireAspNet.Use(GetHangfireServers);
var dretvaId = BackgroundJob.Enqueue(() => dohvat.ApiPriprema());
RecurringJob.AddOrUpdate("Prvi",() => ApiDohvat(),("30 1 * *
*"),TimeZoneInfo.Local);
RecurringJob.RemoveIfExists("Prvi");
```

Prva linija koda odnosi se na odabir servera koji će se rabiti za Hangfire (poziva se metoda koja konfigurira server u kojoj se, među ostalim nalazi i zadnja linija koda). Druga se linija odnosi na kreiranje novog posla te se primjenjuje naredba za stavljanje u red i sam posao izvršava metodu `ApiPriprema()` koja na kraju kreira sve ponavljajuće poslove. Treća linija jest kreiranje tih poslova i radi se s naredbom kreiraj ili ažuriraj koja za prvi parametar (opcionalni) ima ID posla, za drugi parametar prima ono što će posao raditi, za treći prima vremenski izraz kada će se taj posao raditi i zadnji je parametar je vremenska zona (opcionalni). ID je bitan jer sve dretve pozivaju istu metodu, pa bi na kraju bio kreiran samo jedan ponavljajući posao. Vremenski izraz može biti neki vremenski interval (npr. pet minuta ili dan), a u ovom je slučaju kronološki izraz koji znači da će se dretva izvršiti svaki dan u 1:30 ujutro. Vremenska je zona definirana jer bi inače sve dretve bile izvršavane dva sata prije zbog razlike u vremenu između baze i aplikacije. Zadnja je linija brisanje samih ponavljajućih poslova, a oni se brišu jer Hangfire dodatak pamti svaki put kada bi ta dretva bila izvršena, ali nije (ako aplikacija ne radi), te bi na idućem pokretanju aplikacije pokrenula tu dretvu i nastala bi pogreška. Zato se odmah nakon konfiguracije servera, a prije same upotrebe brišu svi poslovi. To je sve vezano uz sam rad dretvi pa će sada će biti opisana sama metoda za dohvat.
Prvo što se radi jest dohvat gradova i kod ispod prikazuje taj dohvat (primjer rada s bazom).

```
List<Gradovi> gradovi = new List<Gradovi>();
 using (BazaVremProgEntities kontekst = new BazaVremProgEntities())
 {
  gradovi = kontekst.Gradovi.ToList();
 }
```

Prva je linija kreiranje liste za gradove, nakon čega slijedi `using` (osigurava uklanjanje objekta na kraju rada) unutar kojeg se kreira objekt tipa `BazaVremProgEntities` koji služi za komunikaciju s bazom (kreiran od entity okvira). Unutar vitičastih zagrada dohvaćaju se

gradovi iz konteksta, i to u obliku liste. Nakon tog dohvata otvara se petlja koja prolazi kroz sve gradove i za svakog dohvaća podatke i sprema ih u bazu. Prvo se radi dohvat s API-ja, što pokazuje sljedeći kod.

```
HttpWebRequest apiRequest = WebRequest.Create("http://api.openweathermap.org/data/2.5/forecast?q=" + grad + ",hr&appid=" + apiKljuc + "&units=metric&lang=hr") as HttpWebRequest;
string apiOdgovor = "";
using (HttpWebResponse response = apiRequest.GetResponse() as HttpWebResponse)
  {
    StreamReader citac = new StreamReader(response.GetResponseStream());
    apiOdgovor = citac.ReadToEnd();
  }
KorjenskiObjekt listaOdgovor = JsonConvert.DeserializeObject<KorjenskiObjekt>(apiOdgovor);
```

Prva linija kreira sam HTTP zahtjev u koji se ubacuju traženi grad i sam API ključ, nakon toga se unutar `using` izraza kreira HTTP odgovor iz kojega se čitaju podatci i stavljaju u varijablu `listaOdgovora` koja je tipa `KorjenskiObjekt` (pomoćni tip za spremanje nakon deserijalizacije JSON-a). Nakon toga slijedi pretvorba iz tog tipa u tip za prognozu koji se može spremiti u bazu, a potom se poziva metoda `ProvjeraISpremanje()` koja prima listu prognoza kao parametar. Prvo se dohvaćaju postojeći podatci iz baze podataka i zatim se, ovisno o broju zapisa, izvršavaju različiti slučajevi. Oni su već bili opisani ispod dijagrama aktivnosti pa će ovdje biti prikazan kod najčešćeg slučaja.

```
else if (progUBazi.Count == listaPrognoza.Count - 1)
 {
  Prognoza zadnjiUListi = listaPrognoza.Last();
  listaPrognoza.RemoveAt(listaPrognoza.Count - 1);
  string bazaLista = string.Join(";", progUBazi);
  string dohvatLista = string.Join(";", listaPrognoza);
  if (!(bazaLista == dohvatLista))
    {
     using (BazaVremProgEntities kontekst = new BazaVremProgEntities())
        {
         foreach (Prognoza red in listaPrognoza)
          {
          kontekst.Prognoza.Add(red);
          kontekst.Entry(red).State = System.Data.Entity.EntityState.Modified;
           }
          kontekst.SaveChanges();
         }
     }
  using (BazaVremProgEntities kontekst = new BazaVremProgEntities())
         {
          kontekst.Prognoza.Add(zadnjiUListi);
          kontekst.SaveChanges();
         }
```

```
}
```

Ovaj slučaj se izvršava ako je broj zapisa u bazi manji za jedan. Prvo se zadnji zapis iz API-ja pohranjuje u varijablu te se nakon toga uklanja iz liste. Liste se pretvaraju u `string` format i uspoređuju, a, ako su različite, lista iz API-ja dodaje se u kontekst (uz promjenu stanja) te se nakon toga kontekst sprema. Na kraju se dodaje i zadnji član (jer je to onaj koji ne postoji u bazi). Kada je ova metoda gotova, vraća se u metodu za dohvat, gdje se još provjerava ima li `i` iz `for` petlje ostatak 0 kada je podijeljen s 56. Ako ima, dretva se zaustavlja na minutu, prije nego što nastavi s dohvatom za idući grad. To se radi zbog ograničenog broja zahtjeva prema API-ju. Kada je gotova petlja, gotova je i sama metoda. To je sve vezano za dohvat, sljedeće će biti opisan prikaz prognoze.

Opis prikaza počet će s kontrolerom za prognozu. On ima dvije metode s istim imenom, jednu za GET zahtjeve, a drugu za POST zahtjeve. Ona za GET praktički samo poziva pogled (popunjava model s praznim podatcima jer nisu dohvaćeni, tu je naravno i model za obrazac). POST metoda dohvaća podatke i zatim vraća pogled, isto kao i u prvoj metodi, kod se svodi na popunjavanje modela (samo što se ovdje model popunjava s podacima dohvaćenima iz baze). Tako će sada biti opisana sama metoda za dohvat za prikaz koja prima podatke o gradu i tipu prognoze kao parametar. Prvo se dohvaća traženi grad i, ako ne postoji, vraća se prazna lista, a, ako postoji, dohvaćaju se podatci (iz baze) i pretvaraju u korektan format (`ModelPrognoza`), te se vraća lista rezultata. U nastavku će biti prikazan kod za dohvat jednodnevne prognoze.

```
List<Prognoza> prognoza = new List<Prognoza>();
if (vrstaPrognoze == "Jednodnevna")
    {
     DateTime datumPocetak = DateTime.Now.AddHours(-2);
     DateTime datumKraj = DateTime.Now.AddHours(22);
     using (BazaVremProgEntities kontekst = new BazaVremProgEntities())
         {
          prognoza = kontekst.Prognoza.Where(c => c.datum >= datumPocetak &&
c.datum <= datumKraj && c.grad == trazeniGrad.id).ToList();
         }
    }
```

Prvo se kreira lista u koju će se spremiti podatci iz baze, zatim dolazi provjera je li riječ o jednodnevnom dohvatu. Unutar samog `if` izraza postavljaju se početni i završni datumi. Početni je smanjen za dva sata zbog razlike u vremenu između aplikacije i API-ja. Isto je tako i za krajnji datum. Podatci se dohvaćaju tako da datum bude veći ili jednak početnom datumu, a manji ili jednak završnom datuma, te da ID bude jednak ID-u traženog grada. Razlika između ovog i peterodnevnog prikaza jest u tome da peterodnevni nema završni datum, nego se dohvaća sve od trenutačnog datuma nadalje. Rezultat se ove metode vraća kontroleru i on popunjava model i poziva pogled. Još jedna stvar koja će biti opisana prije

samog prikaza jest izrada XML-a za graf. Metoda koja za to služi jest `VratiXml()` koja prima listu `ModelPrognoza` kao parametar. Većina posla u vezi s ovom metodom svodi se na smještanje podataka u pomoćnu klasu za XML (`Weatherdata`). Sljedeći kod prikazuje samo kreiranje XML teksta.

```
using (StringWriter stringWriter = new StringWriter() )
 {
  var serializator = new XmlSerializer(klasaXml.GetType());
  serializator.Serialize(stringWriter, klasaXml);
  return "`"+ stringWriter.ToString()+ "`";
  }
```

Prvo se kreira varijabla `stringWriter` u koju će se spremiti tekst XML-a, a zatim se provodi serijalizacija. Klasa po kojoj će XML imati oblik jest `Weatherdata` i varijabla `klasaXml` sadržava sve podatke (varijabla je tipa `Weatherdata`). Na kraju se tekst XML-a vraća u obliku `stringa`.

Što se tiče samog pogleda, u njemu se uporabljuje Razor stroj (eng. engine) kao i kod jednostavne aplikacije, pa se ovdje ne stavlja kod za primjer, ali se opisuje kako pogled radi. Ako ne postoje podatci (jer je korisnik tek prvi put učitao stranicu), tada se prikazuje samo obrazac. Kada korisnik popuni obrazac i zatraži podatke, tada će ti podaci (ako postoje za to mjesto) biti prikazani grafom i u tablicama, a svaka tablica predočuje jedan dan. Način na koji je to napravljeno jest da se prolazi kroz sve podatke i, kada nastane promjena u danu, iscrta se tablica i kreće se na sljedeći dan. Što se tiče grafa, on je preuzet s YR stranice i njegov kod vrlo je kompleksan. Promjene koje su napravljene jesu u tome da ne uzima podatke s te stranice, nego s one koja mu pruža model (XML `string` u modelu), da se ne prikazuje tlak (jer se to ne sprema u bazu), da je graf malo širi i da je većina toga prikazana na hrvatskom. Graf i dalje ne radi savršeno (ne prikazuje sve vrijeme, ne prikazuje dobro vjetar strelicama u nekim slučajevima). To je sve o mogućnosti za prognozu, a u nastavku će biti opisana mogućnost za kartu.

Mogućnost za kartu uglavnom se svodi na JavaScript te će taj dio biti najopširnije opisan. Sam kontroler koji je odgovoran za tu mogućnost jest Home kontroler i koristi se metoda `Karta()` koja samo vraća pogled. Ova mogućnost je ostvarena s pomoću dvaju API-ja (OpenWeather i Google karte). Prvo se kreira sama karta s u kodu ispod navedenim opcijama.

```
var mapOptions = {
        zoom: 9,
        center: new google.maps.LatLng(45.475, 16.78194),
        restriction: {
            latLngBounds: GraniceHrvatska,
            strictBounds: false,
        },
```

```
        minZoom: 8,
        maxZoom: 12
    };
```

Najprije se postavlja opcija za zumiranje, nakon čega se postavlja centar karte na mjesto Kutina (otprilike sredina Hrvatske). Potom slijede restrikcije koje uključuju i samo ograničavanje mape na područje Hrvatske i naposljetku se određuje minimalno i maksimalno približavanje (eng. zoom). Postoji slušatelj događaja kada karta prođe u stanje mirovanja koji inicijalizira dohvat (poziva metodu koja pokreće dohvat). Ta je metoda prikazana u nastavku.

```
var checkIfDataRequested = function () {
    while (gettingData === true) {
        request.abort();
        gettingData = false;
    }
    getCoords();
};
```

Spomenuta metoda dohvaća samo ako je varijabla `gettingData` laž. To služi tomu da, ako se već dohvaćaju podatci, prekida se dohvat podataka i pokreće se novi, a to se radi zato što je došlo do pomaka karte i stari podatci možda više nisu potrebni. Na kraju te metode poziva se sljedeća metoda u seriji metoda za dohvat podataka. Iduća metoda jednostavno dohvaća granice karte i šalje to metodi koja slijedi. Iduća metoda (`getWeather`(), dohvati vrijeme) dohvaća vrijeme preko API-ja, pa je tu trebalo dodati opciju za jezik i sam API ključ. Procesiranje podataka prepušta se idućoj funkciji koja je prikazana u nastavku.

```
var proccessResults = function () {
    console.log(this);
    var results = JSON.parse(this.responseText);
    if (results.list.length > 0) {
        resetData();
        for (var i = 0; i < results.list.length; i++) {
            geoJSON.features.push(jsonToGeoJson(results.list[i]));
        }
        drawIcons(geoJSON);
    }
};
```

Prvo se zapisuje u konzolu, nakon čega se raščlanjuje (eng. parse) rezultat i, ako je on veći od nule, poziva se metoda za brisanje prijašnjih obilježja (eng. feature, u njih se spremaju podatci). Nakon toga se u petlji dodaju nova obilježja s pomoću metode `jsonToGeoJson()` koja zapravo samo stavlja podatke iz JSON formata u korektni format. Tu je trebalo promijeniti da se vrijeme uzima iz opisa vremena (`description`), a ne iz polja koje se zove `main`. I na kraju ove metode s poziva zadnja metoda kod dohvata podataka i ta metoda je `drawIcons()`. Ispod slijedi njezin kod.

```
var drawIcons = function (weather) {
    map.data.addGeoJson(geoJSON);

    gettingData = false;
};
```

Iako spomenuta metoda ima ime nacrtaj ikonice, što na neki način i radi, ona zapravo stavlja sve podatke u podatke karte i stavlja zastavu za dohvat na laž. Jedino što još preostaje za opis u vezi s ovom mogućnošću jest prikazivanje prozora, za što se primjenjuje metoda koja postavlja sadržaj tog prozora i njegovu lokaciju, te ga na kraju naravno prikazuje. To je sve vezano uz mogućnost karta i sada slijedi opis zadnje mogućnosti , a to je statistika.
Zadnja je mogućnost statistika i opis će početi s kontrolerom. Kada korisnik odabere statistiku na navigacijskoj traci, pokreće se prva metoda `Prikaz()`, a postoje dvije, svaka za jednu vrstu zahtjeva. Prva popunjava model dostupnim podatcima (obrazac i liste za prikaz). Primjer je takve liste prikazan u nastavku.

```
statistikaZa = new SelectList(new List<SelectListItem> {
    new SelectListItem {Selected = true, Text="Godina", Value="Godina"},
    new SelectListItem {Selected = false, Text="Mjesec", Value="Mjesec"},
    new SelectListItem {Selected = false, Text="Godina po mjesecima",
Value="GodPoMjesec"}
 }, "Value", "Text", 1);
```

Riječ je o tipu podataka `SelectList` koji se ovdje popunjava članovima koji će biti prikazani u obrascu (obrazac se uglavnom sastoji od takvih lista). Pri kreiranju člana treba odrediti je li član odabran, što će biti prikazano korisniku i koja je vrijednost samog člana (ona se vraća kada korisnik pošalje zahtjev)). Postoje liste za gradove (jer ih nema mnogo), za podatak za koji se traži statistika, gore prikazana lista za razdoblje za koje je statistika (npr. mjesec), vrsta same statistike (npr. maksimum) i godine za koje se može tražiti statistika. Usto, još postoji polje mjesec koje je skriveno i pokazuje se korisniku ako je odabran mjesec na prije prikazanoj listi. Kada korisnik ispravno popuni obrazac i potvrdi ga, tada se pokreće druga metoda `Prikaz()` koja prima model obrasca. Prvo što ta metoda radi je to da poziva određenu metodu za prikaz, ovisno o tome koje je razdoblje korisnik odabrao. Za svako razdoblje postoji metoda i sve primaju sve podatke s obrasca. U nastavku je prikazan dio koda metode za godinu.

```
List<double> listaRezultata = new List<double>();
ModelStatistika prikaz = new ModelStatistika();
prikaz.datumi = new List<string>();
prikaz.rezultati = new List<double>();
prikaz.grad = grad;
prikaz.podatak = podatak;
prikaz.statistikaZa = "Godina";
prikaz.datumi.Add(godina);
for (int i = 1; i <= 12; i++)
```

```
 {
  listaRezultata.Add(dohvat.DohvatStatistike(godina, podatak, statistika,
grad, i.ToString()));
  }
```

Rezultat je uvijek lista tipa duble (iako mjesec i godina imaju samo jedan rezultat, ali godina po mjesecima ima dvanaest). Kreira se model za statistiku i popunjava se podatcima. U petlji se dohvaćaju rezultati za svaki mjesec te se poslije ovisno o statistici (npr. minimum) izračuna statistika za cijelu godinu i sprema pod rezultat. Kod mjeseca nema potrebe za petljom, ni izračunom, a kod godine po mjesecima postoji petlja, ali se ništa ne računa, nego se samo sprema u listu. U nastavku će biti opisana metoda za dohvat kojom se sve metode za prikaz koriste. Ona prima godinu, podatak za koji se traži statistika (npr. tlak), samu statistiku, grad i mjesec (broj od nula do dvanaest, u petljama to je `i`, a kod mjeseca to odabire korisnik). Slijedi prikaz početka te metode.

```
double rezultat=0;
int brojDana = DateTime.DaysInMonth(int.Parse(godina), int.Parse(mjesec));
string pocetakDatum = godina + "-" + mjesec + "-01";
string krajDatum = godina + "-" + mjesec + "-" + brojDana.ToString();
DateTime pocetakTD =DateTime.Parse(pocetakDatum + " 00:00:00");
DateTime krajTD = DateTime.Parse(krajDatum + " 23:00:00");
int idGrad;
bool postoji;
using (BazaVremProgEntities kontekst = new BazaVremProgEntities())
 {
  idGrad = kontekst.Gradovi.Where(l => l.imeGrad == grad).Select(l =>
l.id).FirstOrDefault();
  postoji=kontekst.Statistika.Any(e => e.datum >= pocetakTD && e.datum <=
krajTD && e.grad == idGrad);
  }
if (!postoji)
 {
  ApiDohvatStatistike(idGrad, grad, pocetakDatum, krajDatum);
 }
```

Prvo što je bitno jest saznati broj dana u traženom mjesecu kako bi se mogao napraviti upit. Kreirani su `string` zapisi početka i kraja te se nakon toga kreiraju `DateTime` varijable koje sadržavaju i datum i vrijeme (početak je postavljen na nula sati, a kraj na dvadeset tri, a razlog tomu jest da je zadnji zapis u danu u dvadeset tri sata). U `using` izrazu traži se ID grada na osnovu imena i provjerava se postoji li statistika za taj mjesec (traži se bilo koji zapis jer se uvijek sprema za cijeli mjesec, tako da, ako postoji jedan zapis, trebali bi postojati svi koji su dostupni za taj mjesec). Ako postoje podatci dohvaća se statistika i vraća se rezultat, a, ako ne postoji, pokreće se metoda `ApiDohvatStatistika()` i nakon nje se dohvaća statistika i vraća rezultat. Sama je metoda vrlo slična dohvatu za prognozu,

dohvaća s API-ja, deserijalizira JSON u pomoćnu klasu i u petlji pretvara podatke iz pomoćne klase u onaj tip koji će biti spremljen u bazu. Slijedi prikaz HTTP zahtjev za API.

```
HttpWebRequest apiRequest =
 WebRequest.Create("https://api.meteostat.net/v1/history/hourly?" +
 "station=" +rijecnikGradovi[grad] +"&start=" + pocetak+"&end=" +
kraj+"&time_zone=Europe/Zagreb&time_format=Y-m-d%20H:i&key="+apiKljuc) as
HttpWebRequest;
```

API se zove Meteostat i prvi parametar koji zahtjeva jest broj vremenske stanice koji se dohvaća iz rječnika koji sadržava te brojeve (za ključ ima ime grada), drugi i treći parametar jesu početni i završni datum, a nakon toga slijedi vremenska zona i format datuma, te sam API ključ. Nakon što se na kraju metode podatci spreme u bazu, metoda je gotova i vraća se u metodu za dohvat prikaza, gdje se dohvati statistika i rezultat vraća kontroleru. U kontroler metodi model se popunjava podatcima (obrazac, rezultat, liste za prikaz i rječnik za podatke – sadržava podatke i njihove mjerne jedince) i poziva se pogled. Što se tiče samoga prikaza, on se sastoji od obrasca i tablice za statistiku koja je prikazana samo ako je korisnik već poslao zahtjev za statistikom. Jedino što će biti prikazano kodom jest kreiranje liste s izborom (eng. select list).

```
@Html.DropDownListFor(model => model.forma.grad, Model.gradovi, new {
htmlAttributes = new { @class = "form-control" } })
```

Kao što se vidi, prvo se specificira to u što će se podatak obrasca spremiti pri slanju obrasca, nakon čega slijedi lista kreirana u kontroleru, a naposljetku je CSS klasa. Sama naredba opisuje koji će element kreirati, a to je lista.

7. Zaključak

Zaključak će biti malo više usredotočen na samu aplikaciju i rad u .Net okviru, odnosno kreiranje ASP.NET MVC aplikacije. Rad počinje kratkim opisom svih ključnih jezika za izradu web-aplikacije, a ti su jezici HTML, CSS i JavaScript (povijest, sintaksa i nešto općenito o jeziku). U današnje vrijeme postoje okviri koji imaju unaprijed generirane stilove, HTML dokumente, pa čak i razne JavaScript funkcije. Ti su jezici i dalje prisutni u svim stranicama, ali je upitno koliko je od tog koda zapravo ručno napisano. Potom slijede kratki opisi nekih razvojnih okolina. Nakon toga se uspoređuju C# i neki jezici koji se uporabljuju na strani servera (PHP, Node.js i Python). Svaki od njih ima svoje prednosti i nedostatke i na kraju sve ovisi o tome što točno tražimo od jezika (koji su prioriteti pri izradi web-aplikacije, koliko je vremena na raspolaganju za učenje samog jezika itd.). Nakon te usporedbe slijedi dio o ASP.NET MVC-u (u kojem je napravljena aplikacija). MVC uzorak dijeli se na modele, poglede i kontrolere, od kojih svaki ima svoju ulogu. U modelu najčešće budu spremljeni podatci koji će biti prikazani pogledu, a oni se mogu razlikovati od onih u bazi te mogu postojati modeli koji u sebi imaju druge modele, što omogućuje slanje više modela nekom pogledu. Pogledi su tamo gdje se HTML nalazi (skripte bi, ako je to moguće, trebale biti u Scripts mapi projekta, ali to nije uvijek moguće) i to je prikaz koji će korisnik vidjeti. Postoji shema (eng. layout) u kojoj se nalaze dijelovi stranice koji se uvijek ponavljaju (navigacija, podnožje itd.) i u koje se smještaju pogledi (nakon kreiranja projekta postoji jedna shema, ali programer po potrebi može izraditi nove). Na zadnjem su mjestu kontroleri koji rukuju zahtjevima za koje imaju odgovarajuće metode. Nakon tog dijela prije aplikacije još postoji opis i usporedba nekih okvira za razvoj na strani klijenta. AngularJS je stariji okvir i pun mogućnosti, ali ima i strmu krivulju učenja. React je biblioteka s fokusom na kreiranje korisničkih sučelja, a Vue.js je lagan i jednostavan okvir za razvoj klijentske strane aplikacije.

Aplikacija služi za prikaz vremenske prognoze na području Hrvatske i sastoji se od triju glavnih mogućnosti, a to su prognoza (jednodnevna, peterodnevna), karta koja prikazuje trenutačno vrijeme i jednostavnu statistika za neke hrvatske gradove.

Za prognozu se podatci dohvaćaju s OpenWeather API-ja i dohvaća se prognoza za pet dana. Dohvat se izvodi više puta na dan u određenim dobima dana s pomoću dretava. Sam rad s dretvama na web-aplikaciji nije jednostavan pa se rabi dodatak pod imenom Hangfire koji donekle olakšava taj rad. Ti se podatci spremaju u bazu te, kada korisnik zatraži prognozu, dohvaćaju se i stavljaju u odgovarajući model. Prikaz grafom, odnosno sam graf preuzet je iz YR stranice za prognozu i napravljene su odgovarajuće promjene u kodu kako bi graf radio s podatcima iz baze podataka, a prikaz je u tablici po danima (za svaki dan jedna tablica). Sam Hangfire dodatak ima svoje mane, od kojih je u ovom projektu najviše

smetalo to da, ako se neki posao (dretva) prekine, tada se ona stavlja u red za ponovnu izvedbu, što je bio problem zbog toga jer dohvat podataka traje deset minuta (ograničeni broj zahtjeva), pa je tijekom izrade aplikacije taj dohvat većinu vremena bio isključen da ne bi došlo do prekida dretve.

Sljedeća je mogućnost karta koja je također preuzeta u obliku primjera i nakon toga ograničena na područje Hrvatske i posložena tako da prikazuje informacije na hrvatskom jeziku. Iako zvuči jednostavno, sam je kod relativno kompleksan i trebalo se snaći u njemu kako bi se napravile potrebne izmjene. Iako je karta ograničena na područje Hrvatske, sam dohvat trenutačnoga vremena koji se radi preko OpenWeather API-ja, naravno, dohvaća i gradove okolnih država koje su vidljive na karti jer je Hrvatska neobičnog oblika.

Treća mogućnost jest statistika za koju se dohvat s API-ja vrši po zahtjevu klijenta, a podatci su dostupni od 2000. godine nadalje (osim trenutačne godine). Podatci se dohvaćaju iz drugog API-ja pod nazivom meteostat; nažalost, neki su podatci nepotpuni, a za neke godine ili mjesece čak i ne postoje. S godinama koje su bliže sadašnjosti podatci su obično konzistentniji. Podatci su prikazani u obliku tablice, a dostupne su ove statistike: minimum, maksimum i prosjek za razne vremenske podatke i hrvatske gradove (one koji su veći i imaju u blizini vremensku stanicu).

Sama izrada aplikacije u okviru bila zahtjevna, iako se, kad se pogleda kakva je aplikacija, možda i ne čini tako. Bilo je mnogo problema i prepreka tijekom izrade aplikacije, ali je, nasreću, sam ASP.NET MVC uzorak dobro pokriven na internetu (često se može relativno jednostavno naći rješenje za neki problem). MVC unaprijed kreira mnogo korisnih stvari, kao što je generalna shema (eng. layout) i jedan kontroler i nekoliko akcija (metoda), te, naravno, razne mape koje su bile opisane u radu. Sam dizajn aplikacije odmah je u redu i za sve se uporabljuju razne Bootstrap klase koje su vrlo lijepo složene i dosta često uzimaju u obzir smanjivanje prozora i sam položaj sadržaja. Usto, u okolini je moguće napraviti poglede s raznim predlošcima, što može dosta skratiti i praktički ukloniti rad s HTML-om jer se mnogo toga odmah generira. Također, budući da postoji verifikacija unutar modela, nema ni velike potrebe da se ručno piše JavaScript, ali je na nekoliko mjesta trebalo nešto malo pisati ručno. Sam je okvir definitivno odličan za ljude koji imaju prethodnog iskustva u C# jer se jako puno toga radi u tom jeziku. Naravno, treba naučiti kako se koristiti Razor strojem (eng. engine) kako bi se C# kod mogao pisati unutar pogleda. Sve u svemu, zadovoljan sam okvirom i razvojnom okolinom te samom uporabom jezika jer, da se sve to moralo pisati ručno, bilo bi potrebno mnogo više vremena (naravno, prvo se trebalo naviknuti na uporabu okvira i njegove restrikcije i načela, a i postoje neke naredbe kojima se gotovo nikada ne bi trebalo koristiti u aplikacijama za radnu površinu (eng. desktop)). Najviše je vremena oduzeo rad s Hangfire dodatkom te s komunikacija s API-jima te rad s Razor strojem.

Popis literature

[1] Jeffrey C. Jackson, Web technologies: a computer science perspective, Prentice Hall, 2007.

[2] M. Myers, A Smarter Way to Learn HTML & CSS: Learn it faster. Remember it longer, CreateSpace Independent publishing platform, 2015

[3] E. Freeman i E. Robson, Head First HTML5 Programming, O'Reilly Media, 2011

[4] W3schools (2019) , JavaScript Tutorial, dostupno 05.01.2019 na https://www.w3schools.com/js/default.asp

[5] NetBeans (2019),Documentation, training & support, dostupno 05.01.2019 na https://netbeans.org/index.html

[6] RJ TextEd (2019), RJ TextEd - The Unicode source and text editor, dostupno 05.01.2019 na https://www.rj-texted.se/

[7] Brackets (2019),A modern, open source code editor that understands web design, dostupno 05.01.2019 na http://brackets.io

[8] A. Troelsen i P. Japikse, C# 6.0 and the .NET 4.6 Framework, Apress, 2015

[9] Microsoft (2019), Introduction to ASP.NET Identity, dostupno 05.01.2019 https://docs.Microsoft.com/en-us/aspnet/identity/overview/getting-started/introduction-to-aspnet-identity

[10] J. Duckett, JavaScript and jQuery: Interactive Front-End Web Development, Wiley, 2014

[11] AngularJS (2019),Superheroic JavaScript MVW Framework, dostupno 05.01.2019 na https://angularjs.org/

[12] James Shore (2015),An Unconventional Review of AngularJS, dostupno 05.01.2019 na https://www.letscodeJavaScript.com/v3/blog/2015/01/angular_review

[13] Altexsoft (2018.a), The good and the bad of ReactJS and React Native, dostupno 05.01.2019 na https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-Reactjs-and-React-native/

[14] Altexsoft (2017.b),AngularJS vs Knockout.js vs Vue.js vs Backbone.js Frameworks, dostupno 05.01.2019 na https://www.altexsoft.com/blog/engineering/angularjs-vs-knockout-js-vs-vue-js-vs-backbone-js-which-framework-suits-your-project-best/

[15] Amit Khirale(2018), Comparison between AngularJS vs React in 2018 , dostupno 05.01.2019 na https://dev.to/amitkhirale/comparison-between-angularjs-vs-react-in-2018-1i84

[16] Bernard Kohan (2010), PHP vs ASP.net comparison, dostupno 05.01.2019 na https://www.comentum.com/php-vs-asp.net-comparison.html

[17] Sascha Thattil (2017), Which is better, PHP or ASP.net and why?, dostupno 05.01.2019 na https://www.quora.com/Which-is-better-PHP-or-ASP-NET-and-why

[18] Pratiksha Prasad (2018), PHP 7 vs dot net core: how to make the right choice, dostupno 05.01.2019 na https://yourstory.com/mystory/891449b9cc-php-7-vs-dot-net-core-

[19] Michal Hartwich (2018), Node.js vs APS.NET for enterprise application development services, dostupno 05.01.2019 na https://www.netguru.com/blog/Node.js-vs.-asp.net-for-enterprise-solutions-which-stack-to-choose-for-advanced-web-applications

[20] ilyaigpetrov (2019), Comparison of ASP.NET and Node.js for backend programming, dostupno 05.01.2019 na https://gist.github.com/ilyaigpetrov/f6df3e6f825ae1b5c7e2

[21] SimilarTech (2019), ASP.NET vs Python, dostupno 05.01.2019 na https://www.similartech.com/compare/asp-net-vs-Python

[22] Akash Kava (2016), Between Python and ASP.NET which language is batter for the very first time?, dostupno 05.01.2019 na https://www.quora.com/Between-Python-and-ASP-NET-which-language-is-better-for-the-very-first-time

[23] Jakub Protasiewicz (2018), Python vs C#: comparison of the programming languages, dostupno 05.01.2019 na https://www.netguru.com/blog/Python-vs-c-comparison-of-the-programming-language

[24] K. Siewers Moller (2016), What is the appeal of usign C# for web development rather than other languages, dostupno 07.08.2019 na https://www.quora.com/What-is-the-appeal-of-using-C-for-web-development-rather-than-other-languages

[25] OpenWeather (2019), Weather API, dostupno 27.08.2019 na https://openweathermap.org/api

[26] Meteostat(2019), metostat API, dostupno 27.08.2019 na https://api.meteostat.net/

[27] Hangfire (2019), Documentation, dostupno 27.08.2019 na https://docs.hangfire.io/en/latest/

[28] Rishabh Software (2019), AngularJS vs ReactJS, dostupno 27.08.2019 na https://www.rishabhsoft.com/blog/reactjs-vs-angularjs

[29] Richard York, Beginning CSS: Cascadin Sheets for Web Desing, Wrox, 2007

[30] Shailendra Chauhan (2013) , Understanding detailed architecture of ASP.NET 4.5, dostupno 14.09.2019 na https://www.dotnettricks.com/learn/aspnet/understanding-detailed-architecture-of-aspnet-45

Popis slika

Popis tablica