Smjernice za izradu web aplikacija u TypeScript-u

Žlender, Žan

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:350865

Rights / Prava: Attribution-ShareAlike 3.0 Unported/Imenovanje-Dijeli pod istim uvjetima 3.0

Download date / Datum preuzimanja: **2025-03-23**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ž D I N

Žan Žlender

Smjernice za izradu web aplikacija u TypeScript-u

DIPLOMSKI RAD

Varaždin, 2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Žan Žlender

Matični broj: 0016123668

Studij: Baze podataka i baze znanja

Smjernice za izradu web aplikacija u TypeScript-u

DIPLOMSKI RAD

Mentor:

Prof. dr. sc. Danijel Radošević

Varaždin, rujan 2024.

Žan Žlender

Izjava o izvornosti

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

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

U radu se opisuje programski jezik TypeScript, navode se razlozi radi kojih je stvoren kao nadogradnja za JavaScript te analizom prednosti i nedostataka TypeScript-a argumentira se zašto je prikladan za izradu web aplikacija. Nadalje, istražuje se cjelokupni proces izrade web aplikacije od ideje do finalnog proizvoda. Za svaki od definiranih koraka istražene su trenutno aktualne tehnologije, uspoređene su i dan je prijedlog kada bi se koja trebala, odnosno ne bi trebala, koristiti ovisno o funkcionalnim i sukladno tome tehnološkim zahtjevima aplikacije. Između ostalog naglasak je na brzini razvoja aplikacije, kvaliteti napisanog koda, produktivnosti i iskustvu programera (eng. *developer experience*). Konačno, definirane smjernice korištene su za izradu web aplikacije koja se dotiče trenutno aktualnih tema i/ili rješava problem iz stvarnog svijeta.

Ključne riječi: TypeScript, web, aplikacije, razvoj, programiranje, smjernice, cloud

Sadržaj

1. Uvod 1
2. Metode i tehnike rada 2
3. TypeScript 3
3.1. TypeScript za razvoj web aplikacija 3
3.2. Nedostaci TypeScripta 4
4. Web aplikacije 6
5. Metodologije razvoja web aplikacija 8
6. Koraci razvoja web aplikacije 11
6.1. Ideja 11
6.2. Arhitektura sustava 12
6.2.1. Generalna arhitektura sustava 13
6.2.2. Problemi pri dizajniranju arhitekture 13
6.3. Odabir tehnologija 14
6.3.1. Backend tehnologije 15
6.3.2. Frontend tehnologije 16
6.3.3. Fullstack tehnologije 19
6.3.4. Baza podataka 20
6.3.5. Servisi 21
6.3.6. Verzioniranje koda 22
6.4. Posluživanje aplikacije 25
6.5. Praćenje i bilježenje sustava 31
6.6. Nadogradnja aplikacije 35
6.6.1. CI/CD 35
6.6.2. CI/CD alati 37
7. Kada ne odabrati TypeScript 41
8. Smjernice za izradu web aplikacija 44
9. Izrada web aplikacije prema smjernicama 50
9.1. Primjena smjernica za razvoj aplikacije 50
9.2. Prikaz rada aplikacije 54
9.3. Nadogradnja aplikacije 60
9.4. Utjecaj korištenja smjernice za razvoj web aplikacije 65
10. Zaključak 66
Popis literature 67
Popis slika 71
Popis tablica 72

1. Uvod

Današnje doba većina bi osoba opisala kao žurno, dinamično i nepredvidivo. Spomenute karakteristike poduprijete su konstantnim inovacijama u domeni informatike. Različite web stranice i aplikacije spojile su ljude diljem svijeta, stvorile čitava carstva u domenama internet prodaje i proširila znanje široj populaciji. Svaka osoba sa idejom može krenuti u poslovni svijet u svega par klikova, no kako izgleda pozadina? Kako se može izraditi takva platforma? Ovim radom želi se upoznati čitatelja sa procesom razvoja web aplikacije pomoću TypeScripta te mu dati savjete kako ući u jedan mali dio tog svijeta, svijeta web razvoja.

Trenutno stanje razvoja web aplikacija veoma je burno. Naočigled se čini kao da se svaki mjesec izrađuje novi razvojni okvir, dešavaju se konstantne promjene u savjetima i najboljim praksama koje pružaju i najutjecajniji programeri, uveliko se mijenja način razmišljanja cijele programerske zajednice, a to je tek početak. Iz tog razloga rasprostranjeno je mišljenje da je danas teže krenuti sa web razvojem, a još teže kvalitetno razvijati, no je li to stvarno tako? Radi svega navedenoga, ponekad može biti veoma kompleksno odabrati ispravan način za izradu web aplikacije za svoju najnoviju ideju ili start-up. U ovome radu nastojati će se objektivno analizirati stvarno stanje web razvoja danas, odrediti korake i informacije potrebne za stvaranje funkcionalne web aplikacije te pružiti detaljan uvid u trenutno popularne tehnologije i/ili servise koji predstavljaju temelj za izradu gotovo svake aplikacije.

Ipak, kako je obujam web razvoja prevelik, u ovome radu neće se moći obraditi svi detalji o svim tehnologijama i najboljim praksama, kako je za svaku tehnologiju moguće zasebno napisati čitav rad. Cilj ovog rada jest kreirati smjernice za razvoj web aplikacija pomoću TypeScripta koje naglasak stavljaju na jednostavnost implementacije, niske troškove, iskustvo tokom programiranja i modularnost funkcionalnosti. Samim time naglasak neće biti na samu implementaciju već na generalni proces razvoja web aplikacije sa TypeScriptom kao sponom koja povezuje sve druge aspekte razvoja.

2. Metode i tehnike rada

Temelj ovog rada predstavlja presjek informacija iz literature i empirijske analize iz struke. Prema zadanim kriterijima odabrani su stručni članci, knjige te materijali i smjernice od strana kreatora spominjanih tehnologija, utjecajnih programera i vodećih inženjera današnjice.

Pristup prilikom pisanja ovog rada bio je analitički i empirijski. Proučavanjem i analiziranjem spomenutih izvora nastojalo se doprijeti do teorijske podloge zašto je prikladno koristiti TypeScript kao polaznu točku za razvoj web aplikacija i kako odabrati potrebne popratne tehnologije. Zatim slijedi empirijski dio, analiza tehnologija jest temeljena na teorijskoj podlozi ali i osobnom iskustvu stečenim akademskim obrazovanjem i poslovnom izobrazbom kao web programera, u čiju sferu pripada i cjelokupan proces kreiranje web aplikacije.

Za izradu smjernica za razvoj web aplikacija u TypeScriptu definirani su određeni koraci u razvoju web aplikacija te potrebnih tehnologija, servisa i razvojnih okvira u cjelokupnom procesu. Tehnologije i pristup odabrani su na način da sadrže rješenje za velik broj potencijalnih zahtjeva aplikacije s naglaskom na modularnost, brzinu razvoja, jednostavnost korištenja, skalabilnost aplikacije i dobro iskustvo programiranja. Polazna motivacija jest svakom programeru omogućiti da od početne ideje za web aplikaciju ima definiran skup tehnologija potrebnih za izradu iste.

3. TypeScript

Prije nego što se detaljnije analizira TypeScript, nužno je započeti od prethodnika TypeScripta, odnosno od JavaScripta, često smatranim najpoznatijim programskih jezikom u svijetu [52] . JavaScript pokreće većinu sadržaja koji koristimo putem weba, ono je skriptni jezik koji omogućuje implementaciju kompleksnih značajki na web stranicama [53].

JavaScript često se spominje u kontekstu web stranica, no ovaj programski jezik pokreće i mnoštvo aplikacija drugačije namjene. Kao što će biti prikazano u narednim poglavljima, JavaScript može pokretati web poslužitelje, RESTful i druge API servise te gotovo svakakve druge tipove programa koje je moguće zamisliti. Iako se JavaScript unapređivao iz godine u godinu i dalje nije idealan jezik, što je omogućilo pojavu nadogradnji na nj, kao što je TypeScript. TypeScript je programski jezik i tzv. superskript JavaScripta. Drugim riječima, TypeScript je programski jezik sa strogo definiranim tipovima koji je izrađen na temelju JavaScripta [52]. Kako JavaScript sam za sebe ne posjeduje definiciju tipova varijabli koje se koriste, TypeScript uveliko olakšava izradu većih aplikacija ponajviše jer pojednostavljuje razvoj aplikacija i smanjuje mogućnost grešaka u finalnom proizvodu. TypeScript se izvodi tokom samog programiranja (eng. *during runtime*) čime se unutar integriranog programskog okruženja dobivaju rana upozorenja, dokumentacija metoda, mogućnost automatske nadopune koda i mnoge druge pogodnosti koje će biti definirane u narednim poglavljima.

3.1. TypeScript za razvoj web aplikacija

Kao što je već bilo spominjano u prethodnom poglavlju, razvoj modernih aplikacija karakterizira brzina razvoja i promjena funkcionalnosti ovisno o potrebama korisnika. U tome slučaju jedna od velikih prednosti TypeScripta u odnosu na JavaScript jest sigurnost koju programeri dobivaju tokom pisanja koda. Statično definirani tipovi omogućuju otkrivanje grešaka tokom izvođenja (eng. *during runtime*) [52]. Drugim riječima, alati za uređivanje programskog koda javljaju greške kako se događaju tokom pisanja, umjesto da budu otkrivane tokom izgradnje aplikacije (eng. *during build time*) ili još gore u produkcijskom okruženju. Alati za uređivanje programskog koda također omogućuju automatsko dovršavanje koda, prikaz potrebnih parametara kao i povratne tipove funkcija koji pojednostavljuju proces razvoja. Dodatno, samo statično definiranje tipova olakšava nadogradnju i održavanje aplikacija, kako pojedini dijelovi aplikacije moraju slijediti unaprijed definirane ugovore, odnosno tipove. TypeScript je također relativno jednostavan za korištenje u usporedbi s drugim sličnim jezicima, čime ostvaruje kvalitetno iskustvo tokom programiranja i ubrzava cjelokupan proces razvoja. Posljednja prednost je kompatibilnost s modernim web preglednicima. Kako je

TypeScript superskript JavaScriptu, sav kod napisan u TypeScriptu pretvara se u JavaScript što omogućuje da se može izvoditi u svakom web pregledniku [54, 55].

3.2. Nedostaci TypeScripta

Iako su do sada bile definirane velike prednosti koje donosi korištenje TypeScripta, ni ovaj programski jezik nije idealan. Generalne zamjerke TypeScripta dotiču se performansi gdje se često uspoređuje s nekim od performantnijih jezika poput C++a, Golanga i sl. [56]. No ovaj slučaj očit je tek kada aplikacije budu jako korištene ili zahtijevaju izrazito visoke performanse i/ili manipulacije podacima. Problemi koji se javljaju u takvim trenucima također uključuju curenje memorije (eng. *memory leak*) kao i probleme sa skalabilnošću kako TypeScript, odnosno JavaScript, radi sa samo jednom dretvom (eng. *single-thread*). Ipak, danas je većina tih problema riješena, najvećim dijelom radi pojave bezserverske arhitekture (eng. *serverless architecture*) koja će biti objašnjena u narednim poglavljima [17].

Nadalje, iako statični tipovi olakšavaju pisanje i održavanje koda, potrebno je uvijek definirati sve tipove, što ponekad zahtijeva duboko znanje TypeScripta. Jednostavni tipovi poput: integer, string, boolean; nisu zahtjevni, no implementacija može postati zamršenija kada su potrebni generični i/ili kompleksni tipovi [56]. Primjer jednog takvog tipa vidljiv je u isječku koda 1, koji je preuzet iz popularne biblioteke tRPC, korištene za pozivanje statično definiranih funkcija na serverskoj strani.

Programski kod 1. Isječak tipa iz biblioteke tRPC (Izvor: [32])

```
type DecoratedProcedureRecord<
  TRouter extends AnyRouter,
  TProcedures extends ProcedureRouterRecord,
> = {
  [TKey in keyof TProcedures]: TProcedures[TKey] extends AnyRouter
    ? DecoratedProcedureRecord<TRouter,
TProcedures[TKey]['_def']['record']>
    : TProcedures[TKey] extends AnyProcedure
    ? DecorateProcedure<TRouter['_def']['_config'], TProcedures[TKey]>
    : never;
};
```

Isječak koda 1, pokazuje potreban tip za statično definirane funkcije koje se mogu pozivati i izvoditi na serveru, pri čemu su prilikom pozivanja funkcije povratni tip podataka te funkcije definirane od strane programera. Za potrebe ovog primjera nije potrebno detaljno objasniti sav

kod, no kao što je uočljivo iz priloženog nije tako jednostavno protumačiti što točno ovaj dio koda radi.

Posljednja zamjerka koja se pridodaje TypeScriptu jest problem u performansama tokom programiranja kod većih projekata. TypeScript tokom razvoja mora u realnom vremenu (eng. *on the fly*) zaključiti koji se tipovi koriste u danom trenutku, ovisno o unosu programera. Primjerice, ako imamo definirana dva tipa, pri čemu je jedan ugniježđen unutar drugog, TypeScript mora prolaziti sve tipove kako bi znao zaključiti finalan tip. Slijedno tome, što je veći projekt potrebno je više zaključivanja, što kod slabijih računala ubrzo može dovesti do smanjenih performansi, zamrzavanja uređivača koda i sl.

4. Web aplikacije

Prije no što budu definirani koraci izrade web aplikacije, prvo je potrebno definirati što ona jest. Generalno, web aplikacije definiraju se kao “program koji se pokreće u web pregledniku, pri čemu koristi kombinaciju serverskog jezika (Java, Python, itd.) za odrađivanje poslovne logike, pohrane itd., te klijentski jezik (HTML i JavaScript) za prezentaciju” [16]. Također, valja napomenuti razliku između web aplikacija i web stranica. Iako postoje mnoge zajedničke karakteristike među njima, najveća se razlika svodi na interaktivnost. Samim time, ova dva termina često se koriste sinonimno, no jedna stvar koja se dade pretpostaviti jest da izrada web aplikacija zahtijeva korištenje kompleksnijih, i često naprednijih, tehnologija.

Arhitektura web aplikacija ovisi o zahtjevima same aplikacije, ali generalno uključuje sljedeće stavke: klijentska strana (eng. *frontend*), serverska strana (eng. *backend*), bazu podataka, sučelje za programiranje aplikacije (eng. *Application Programming Interface*), u nastavku API, i web server. U bazu podataka spremaju se podaci ključni za funkcioniranje aplikacije i prikaz podataka na stranici. Serverska strana podrazumijeva program koji je pokrenut na serveru te koji služi za dohvaćanje i umetanje podataka u bazu, odrađivanje zahtjeva od strane klijenta, obradu podataka i sl. API je posrednik između klijentske i serverske strane koji omogućuje međusobnu komunikaciju. Samim time izolirana je poslovna logika aplikacije na serversku stranu što sustav čini sigurnijim i otežava zlonamjernim činiteljima da neispravno koriste aplikaciju. Klijentska strana služi kao prezentacijski sloj koji korisniku prikazuje podatke u smislenom obliku te omogućuje korištenje same aplikacije. Posljednja stavka, web server, služi za samo pokretanje i posluživanje aplikacije diljem svijeta. Promatrajući realnu interakciju spomenutih stavki, primjer može biti web aplikacija za spremanje bilježaka. Korisnik može koristiti samu stranicu gdje može unositi i uređivati bilješke. Kada korisnik želi kreirati novu bilješku zahtjev se šalje na API koji je pokrenut na serverskoj strani. Ovisno o obliku zahtjeva, serverska aplikacija određuje što joj je činiti. U spomenutom primjeru, povezuje se s bazom podataka i unosi novu bilješku. Jednako tako, kod prikaza podataka ponovo se šalje zahtjev na API koja tada dohvaća sve bilješke za trenutnog korisnika i vraća te podatke na klijentsku stranu za prikaz.

Postoje različite vrste web aplikacija neke od kojih su bile spomenute u početnom dijelu ovog poglavlja. Najčešće, web aplikacije dijele se u 4 kategorije prema funkcionalnosti i kompleksnosti:

- Statične web aplikacije
- Dinamičke web aplikacije
- Jednostranične aplikacije (eng. *Single-Page applications - SPAs*)

- Progresivne web aplikacije (eng. *Progressive Web applications - PWAs*)

Statične web stranice, kao što naziv dade zamijetiti, prikazuju samo statične podatke. U ovoj vrsti web stranicu iz danog seta podataka kreiraju se gotovi HTML dokumenti koji imaju sav sadržaj stranice umetnut u sam dokument. Iako statične web stranice mogu imati interakciju u pogledu animacija i akcija koje korisnik može poduzeti, podaci se nigdje ne spremaju, ne postoji backend niti API za komunikaciju s njime. Dinamičke web aplikacije pak uključuju upravo to, komunikaciju s backendom, pohranu i prikaz ažurnih podataka i sl. Ovakve web aplikacije često imaju predloške gdje se prilikom pristupa stranici od strane korisnika ažurni podaci preuzimaju iz baze podataka i ubacuju u predložak, što korisniku omogućuje pregled i ažuriranje aktualnih podataka. Samim time, dinamičke aplikacije uključuju backend i neki način pohrane podataka, primjerice u bazu podatka. Nadalje, jedna podvrsta dinamičkih aplikacija jesu jednostranične aplikacije koje koriste neki programski jezik, primjerice JavaScript, kako bi dohvatile i umetnule podatke u HTML dokument. Za razliku od klasičnih dinamičkih stranica gdje u pravilu svaka putanja odgovara zasebnom HTML dokumentu, jednostranične aplikacije koriste JavaScript kako bi simulirale otvaranje stranica, iako se sve odvija na jednoj, istoj stranici. Primjer jednostraničnih aplikacija jesu React aplikacije. React je biblioteka koja se povezuje na temeljni (eng. *root*) element stranice i dinamički ubacuje sav HTML u spomenuti element. Klikom na navigacijski element, pri čemu je korisnika potrebno preusmjeriti na novu putanju, sav sadržaju u temeljnom elementu se obriše te se ponovo ubacuje novo generirani HTML. Posljednja vrsta jesu progresivne web aplikacije koje mogu biti podvrsta bilo koje od 3 prethodno spominjane vrste. Ipak, funkcionalnosti koje razlikuju progresivne web aplikacije od ostalih jesu mogućnost rada bez internetske veze, korištenje obavijesti operacijskog sustava na kojem se nalaze i jednostavnije korištenje ugrađenih mogućnosti uređaja kao što su kamera, GPS i sl.

Web aplikacije izmijenile su način na koji korisnici koriste aplikacije. Svaka web aplikacija, ako je tako podešena, može biti pristupana od bilo koje osobe koja ima pristup internetu i web pregledniku, neovisno o uređaju. Godinama razvoja metodologija web aplikacija razvijeni su brojni jezici i načini izrade istih. Pojednostavljeno je stvaranje interaktivnih i skalabilnih aplikacija koje poslužuju milijarde korisnika svaki dan. U narednim poglavljima biti će detaljnije objašnjen proces stvaranje web aplikacije, odluke koje je potrebno donositi prije i prilikom razvoja te kako najadekvatnije odabrati ispravne tehnologije s ciljem olakšavanja cjelokupnog procesa razvoja.

5. Metodologije razvoja web aplikacija

Kako su aplikacije postajale sve opširnije, povećavali su se i zahtjevi te je postalo neophodno razviti procese koji će doprinijeti čim efikasnijem i kvalitetnijem razvoju. Tako su razvijene neke od danas najkorištenijih metodologija za razvoj programa: Agilni razvoj, Kanban i Scrum [60]. Prema Janesu i Succiju „agilna metoda uvela je novu perspektivu u razvoj programa: snažan fokus na agilnost i mogućnost prilagodbe na razne zahtjeve okoline“ [57]. Karakterizirana iterativnim procesom sa naglaskom na prilagodljivost, sudjelovanje i zadovoljstvo korisnika, agilna metoda odvija se u manjim koracima gdje se isporučuje dio programa, naspram isporuke čitavog finalnog proizvoda odjednom. Pojedine iteracije nazivaju se *sprintevi* koji se dogovaraju unaprijed na sastancima i uglavnom traju dva do četiri tjedna. Samim time omogućuje se dobivanje povratnih informacija od strane korisnika čime se može dobiti uvid kako unaprijediti kvalitetu aplikacije. Kako agilni pristup donosi prednosti u razvoju, na temelju njega razvijane su druge metodologije, primjerice Scrum metodologija. Scrum nadodaje procese za bolje upravljanje projektom i timom, pa tako „Scrum tim nadgledavaju Scrum voditelj (eng. *Scrum master*) i Vlasnik proizvoda (eng. *Product owner*).“ [59]. Dnevni sastanci sastavni su dio ove metodologije sa ciljem praćenja odrađenih zadataka i planiranja nadolazećih. Uvidjevši velik utrošak vremena na količinu procesa koja se odvijala, razvijena je Kanban metodologija „čija je glavna ideja vizualizacija toka rada“ [59]. Za razliku od Scruma, Kanban smanjuje kompleksnost tako da ne razlikuje uloge u grupi, moguće je mijenjati zadatke tokom sprinta, a planirani posao se nije ograničen unutar sprinteva.

Iako postoje brojne prednosti korištenja spomenutih metodologija, postoje i zamjerke. Primjerice, Janes i Succi napominju kako „agilni pristup više nije smatran najboljim rješenjem za sve...“ te zaključuju da „za izbjegavanje da agilne metode postanu nepotrebne, smatramo da je potrebno biti upoznat sa taktikama razvoja programa, znati kada što primijeniti ovisno o prednostima i nedostacima“. Nadalje, poznati informatičar Meyer u svojoj knjizi piše kako „neke agilne tehnike koje su preporučene jednostavno su netočne, kontradiktorne dokazanim pravilima dobrog programskog inženjerstva...“ [61]. Štoviše, autor agilne metodologije, Robert Martin, u jednom intervjuu izjavio je: „Poruka agilne metode postala je toliko izokrenuta, zategnuta i izopačena da sam napisao novu knjigu sa ciljem da ponovo pokrenem ovu diskusiju...“ [62].

Sa svime spomenutim dade se zaključiti kako su agilne metode veoma korisne pri razvoju aplikacija no postoje manjkavosti u njihovom procesu. Svaka metodologija apstraktno propisuje što valja činiti, iako to omogućuje primjenu na širi spektar zahtjeva vrsta razvoja, za svaku pojedinu aplikaciju nedostaju koraci koji opisuju proces same odluke o načinu implementacije. Jednako kako su Scrum i Kanban nastojali unaprijediti Agilni pristup, ovaj rad

predlaže generalne korake u razvoju aplikacija koji konkretiziraju procese u agilnim metodologijama i prilagođeni su za korištenje uz TypeScript. Konkretno, za polaznu točku određen je Kanban radi svoje jednostavnosti i uspješnosti u industriji. Prema posljednjem istraživanju Kanban Universityja o korištenosti Kanbana, ustanovljeno je da čak 86% ispitanih poduzeća u nekoj mjeri koristi, ili planira koristiti Kanban unutar narednih 12 mjeseci [63]. Postoji svega šest tehnika implementacije Kanbana:

1. Vizualiziraj tokove rada
2. Ograniči posao u obradi
3. Upravljaj tokom projekta
4. Osiguraj definirane procese
5. Imaj mogućnost davanja povratnih informacija
6. Unapređuj zajedno

Vizualizacija tokova rada osigurava preglednost cijelog procesa a postiže se Kanban pločama. Svaka ploča sastoji se od stupaca jedinstvenog značenja, primjerice stupci: „potrebno odraditi“, „u izradi“, „završeno“. Zadaci se stavljaju u stupac tablice ovisno u kojoj je fazi izrade. Preglednost zadataka omogućuje programerima da se fokusiraju na zadatke koje moraju i tako ograniče posao u izradi. Nadalje, ploča omogućava otkrivanje potencijalnih prepreka i zastoja te tako olakšava upravljanje tokom projekta. Kako se zahtjevi mogu mijenjati zadužene osobe u timu kreiraju nove zadatke, a svi članovi mogu davati povratne informacije i otvoreno pokrenuti diskusiju o pojedinoj temi, čime svi doprinose unapređenju projekta i pojedinaca [64].

Kada se agilne metodologije razmatraju u kontekstu životnog ciklusa razvoja web aplikacija (eng. *Web Development Life Cycle - WDLC*) mogu se uočiti preklapanja. WDLC sastoji se od 6 koraka [65]:

1. Prikupljanje informacija (eng. *Gathering relevant information*)
2. Planiranje: Pregled stranice i žičani model (eng. Planning: Sitemap and Wireframe)
3. Dizajn: Raspored (eng. Design: Layout)
4. Razvoj (eng. *Development*)
5. Testiranje, provjera i postavljanje aplikacije u pogon (eng. *Testing, Review and launch*)
6. Prikupljanje relevantnih informacija (eng. *Gathering relevant information*)

Slično agilnim metodologijama, životni ciklus razvoja aplikacije iterativan je i cikličan. Prvo je potrebno prikupiti informacije bitne za sam projekt, što klijent očekuje, koji zahtjevi moraju biti ispunjeni, moraju li se podaci moći pohranjivati i sl. Prema prikupljenim informacijama kreiraju se žičani model i pregled stranica web aplikacije, što predstavlja temelj za izradu dizajna same

aplikacije. Kada su definirani svi spomenuti koraci moguće je krenuti sa razvojem aplikacije, testiranjem te prikupljanjem informacija gotovog dijela aplikacije kako bi se unaprijedila. U prethodno spomenutom Kanban modelu, svaki od koraka zaseban je zadatak koji je potrebno izvršiti. Samim time može se zaključiti agilni pristup svojom prilagodljivošću odgovara razvoju aplikacije, kako i ono često zahtjeva redovite promjene. Ipak, dijeli slične probleme definirane u prethodnom dijelu ovog poglavlja, generalizira se razvoj. Ne postoje smjernice specifično za razvoj web aplikacija koje odgovaraju na pitanje kako i što odabrati, već samo da je potrebno „prikupiti informacije“, „planirati“, „razvijati“, „testirati“ i „iterirati ovisno o prikupljenim informacijama“.

Kako je tema ovog rada ograničena na razvoj web aplikacija u TypeScriptu moguće je pobliže definirati procese koji će uzeti u obzir TypeScript ekosustav, trenutno aktivne tehnologije i najbolje prakse u industriji. U nastavku biti će definirani pojedini koraci razvoja aplikacije kao dodatak Kanban metodologiji prateći ciklus razvoja web aplikacije od same ideje sve do konačnog proizvoda, sa ciljem pojednostavljivanja i pokretanja diskusije o implementaciji aplikacije.

6. Koraci razvoja web aplikacije

Tokom povijesti razvoja web aplikacija razvijani su različiti razvojni okviri i standardi za njihovu izradu. Ipak, problem kod definiranja generalnog pristupa jest ta da ne postoji jedno idealno rješenje, kako se sve može promijeniti ovisno o zahtjevima aplikacije [57]. Iz tog razloga, ovaj rad fokusira se na izradu same aplikacije, odnosno kako najbrže i najadekvatnije stvoriti minimalno održiv proizvod (eng. *Minimum viable product - MVP*), a ne na samu implementaciju. Pružiti će se uvid u problematiku razvoja web aplikacija, najčešće funkcionalnosti koje je potrebno implementirati te kako najbolje pristupiti problemu dizajniranja cijelog sustava. Valja napomenuti kako se koraci definirani u ovome poglavlju temeljeni na sintezi najboljih praksi pronađenih u industriji i empirijske analize. Radi svoje promjenjivosti, ovisno o najnovijim saznanjima, najaktualnije i najbolje prakse uglavnom su dijeljenje online od strane poduzeća ili drugih programera. U narednim poglavljima razraditi će se koraci u razvoju web aplikacije sa fokusom na različite aspekte razvoja koje je potrebno uzeti u obzir kao i neke od preporučenih tehnologija za svaku od definiranih domena, počevši od same ideje.

6.1. Ideja

Najveće i najuspješnije web aplikacije današnjice krenule su od iste početne točke, od ideje. U ovome koraku potrebno je dobro promisliti kako realizirati ideju, pod time se ne podrazumijevaju tehničke karakteristike aplikacije, već što je cilj aplikacije kao i same funkcionalnosti aplikacije.

Ovaj dio razvoja aplikacije često se naziva izrada (ne)funkcionalnih, ili korisničkih, zahtjeva. Iako postoje određene strategije kako najbolje odraditi ovaj korak, za početak dovoljno je zamisliti se kao korisnika te aplikacije te definirati kako bi takav korisnik mogao koristiti aplikaciju. Tablica 1 primjer je jednog od načina definiranja korisničkih zahtjeva. Tablica je podijeljena u 4 stupca, stupac “Oznaka” predstavlja jedinstvenu šifru koja služi jednostavnijem praćenju zahtjeva. Stupac “Naziv” sadrži kratki naziv željene funkcionalnosti, dok stupac “Opis” sadrži opis te funkcionalnosti. Posljednji stupac “Bitno” je opcionalan, no služi za definiranje redoslijeda izrade funkcionalnosti prema važnosti, primjerice na skali od 1 do 5, pri čemu je 1 najbitniji, a 5 najmanje bitan zahtjev.

Tablica 1. Primjer definiranja funkcionalnih zahtjeva

Oznaka	**Naziv**	**Opis**	**Bitno**
F01	Prijava	Kao korisnik želim se moći prijaviti u aplikaciju.	1
F02	Pregled objava	Kao korisnik želim moći vidjeti popis svih X.	2
…	…	…	…

Dobro definirani korisnički zahtjevi predstavljaju temelj izrade kvalitetne aplikacije, iako se oni ponekad tokom razvoja aplikacije mogu mijenjati. Valja napomenuti kako bi se funkcionalnosti trebale dodavati a manje mijenjati. Ipak, ovdje treba razlikovati 2 veoma bitna i različita slučaja, da li se aplikacija izrađuje za nekog klijenta ili za vlastite svrhe. Ako se aplikacija izrađuje za klijenta definirani korisnički zahtjevi često predstavljaju pravni dokument koji osigurava izradu konkretno dogovorene aplikacije, dok se u slučaju vlastite izrade zahtjevi jednostavnije i bez posljedica mogu mijenjati.

Konkretizirana ideja predstavlja temelj koji će odrediti pravac u kojem će se aplikacija kasnije razvijati. Jednom definirani (ne)funkcionalni zahtjevi omogućuju odlučivanje adekvatnih tehnologija i servisa, koji predstavljaju početnu točku dizajna arhitekture cijelog sustava, što će biti detaljnije razrađeno u narednom poglavlju.

6.2. Arhitektura sustava

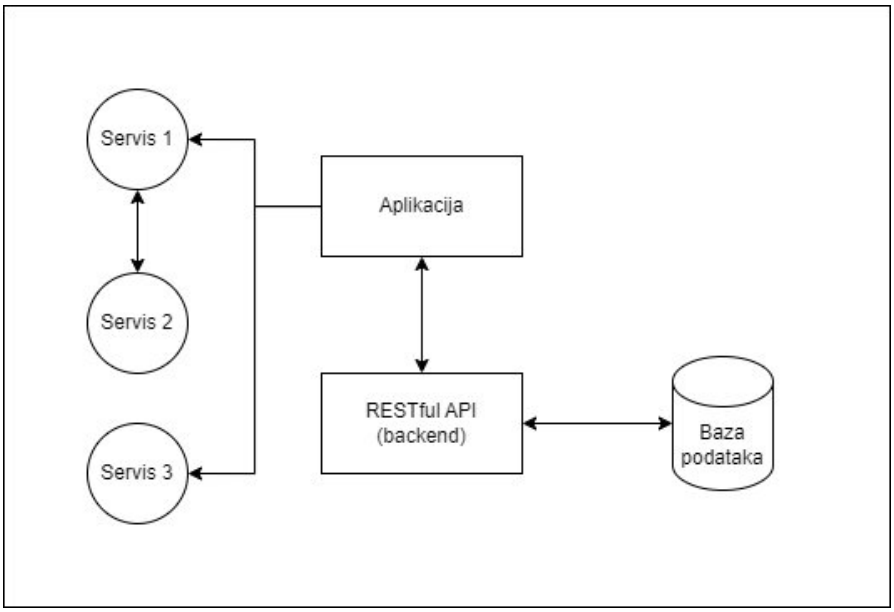
Kada je ideja razrađena potrebno je osmisliti generalnu arhitekturu sustava, što prema Faisandieru znači “definirati niz detaljnih rješenja, temeljenih na principima, konceptima i svojstvima koji su logički povezani i konzistentni”, pri čemu moraju “u najboljoj mogućoj mjeri zadovoljiti zahtjeve sustava” [1]. Drugim riječima, potrebno je imati na umu sve sastavnice i zahtjeve aplikacije te sukladno tome odabrati najbolje servise, tehnologije, način rada i organizaciju kojom će se moći izraditi željena aplikacija.

U kontekstu web aplikacija postoji nekoliko generalnih pristupa arhitekturi, MVC (eng. *Model View Controller*), MVVM (eng. *Model View Viewmodel*), mikroservisna arhitektura i sl. Ipak, svi ovi pristupi ovise o zadanom problemu i pristupu određenom problemu koji aplikacija rješava. Nadalje, u većini slučajeva pri programiranju sa TypeScriptom programeri se odlučuju za određene tehnologije i/ili razvojne okvire koji implementiraju jedan ili više od spomenutih pristupa, a sami programeri ne razmatraju toliko tu problematiku.

6.2.1. Generalna arhitektura sustava

Pojam generalna arhitektura sustava ne dotiče se striktno dizajniranja potpune i završne arhitekture programskog dijela aplikacije, već širokog pogleda na aplikaciju (eng. *high-level overview*). U ovome koraku najbitnije je ugrubo odabrati generalne tehnologije i servise koji će se koristiti, a ne nužno sve detaljno razraditi [16].

Na slici 1. prikazan je primjer pojednostavljene arhitekture sustava koja se sastoji od grafičkog sučelja, odnosno same aplikacije, backend servisa koji služi za komunikaciju sa bazom podataka te 3 dodatna servisa koji komuniciraju sa aplikacijom ili međusobno. Ovaj primjer je proizvoljan i ne postoji realan problem koji rješava, već služi kao prikaz mogućeg dizajna generalne arhitekture sustava.



Slika 1. Pojednostavljeni primjer širokog pogleda arhitekture sustava (Izvor: Vlastita izrada)

Iako se naizgled može činiti jednostavno, potrebno je imati širok spektar znanja u domeni razvoja web aplikacija kako bi se adekvatno mogla izraditi arhitektura sustava. Potrebno je biti upoznat sa dobrim i lošim stranama odabranih tehnologija i servisa kao i njihovih ograničenja. U nastavku će biti objašnjeni neki od problema na koje se može naići pri dizajniranju arhitekture sustava, aspekte koje je potrebno uzeti u obzir kao i neka rješenja koja se danas koriste.

6.2.2. Problemi pri dizajniranju arhitekture

Dizajniranje arhitekture sustava suštinski je problematično i zahtjeva razmatranje prednosti nedostataka svakog dijela sustava. Prva kategorija takvih problema proizlazi iz funkcionalnih zahtjeva aplikacije. Primjerice, ako je jedan od zahtjeva mogućnost prijave

korisnika, potrebno je imati bazu podataka u koju će se spremati podaci o korisniku, potrebno je sučelje za prijavu kao i backend koji će služiti kao spona između sučelja i baze podataka. Kao što je uočljivo iz definiranog primjera, zahtjevi aplikacije određuju koje tehnologije i servisi su potrebni za implementaciju, što pak uvjetuje kako će izgledati arhitektura sustava. Druga kategorija problema proizlazi iz nefunkcionalnih zahtjeva, kao što je: broj korisnika, brzina izvođenja, troškovi i sl [33].

Kako dizajniranje arhitekture sustava utječe na sve naredne korake, poprilično je bitno na početku odabrati kvalitetne tehnologije, kako kasnije ne bi uzrokovali velike probleme, pa čak i nemogućnost realizacije aplikacije. Stoga je prilikom dizajniranja arhitekture potrebno je uzeti u obzir mnoge aspekte, samo neki od kojih su:

- predviđeni broj korisnika aplikacije
- broj programera koji će raditi u timu
- brzina izvođenja aplikacije, odnosno zahtjevi glede performansi
- skalabilnost sustava
- predviđeni troškovi
- funkcionalni i nefunkcionalni zahtjevi
- veličina projekta
- praćenje, nadogradnja i analitika sustava

Odlučivanje o idealnom rješenju nije jednostavno, stoga će se u narednom poglavlju objasniti način odabira tehnologija sa naglaskom na rješavanje prethodno definiranih problema.

6.3. Odabir tehnologija

Kao što je više puta bilo spominjano u ovome radu, odabir tehnologija uvelike određuje smjer projekta, pa samim time i budućnost web aplikacije. U nastavku će se detaljnije razjasniti koje vrste tehnologija generalno svaka aplikacija sadrži te kako odabrati najbolje tehnologije. Polazna točka biti će TypeScript zbog svih prednosti koje su bile navedene u poglavlju 3. Stoga će sve odabrane tehnologije i servisi biti takvi da odgovaraju korištenju uz TypeScript i/ili JavaScript. Važno je napomenuti prije no što će biti analizirane prednosti i nedostaci odabranih tehnologija jest da sve što će biti spominjano u nastavku predstavlja samo polaznu točku za razvoj aplikacije. Svaki zahtjev aplikacije može potpuno promijeniti pogled na adekvatne tehnologije i njihovu implementaciju. Iz tog razloga ne valja previše razmišljati o samim tehnologija već odabrati nešto i krenuti sa izradom te rješavati probleme kako naiđu.

6.3.1. Backend tehnologije

Backend strana često se koristi sinonimno sa serverom, to je sav kod koji se neće izvesti na web pregledniku korisnika aplikacije. Drugim riječima, “backend je fokusiran na serversku stranu web aplikacija. Koriste tehničke sposobnosti kako bi osigurali strukturu i generalnu funkcionalnost te omogućili frontendu da postoji” [2]. U kontekstu TypeScripta kao backend tehnologije postoji nekoliko razvojnih okvira i biblioteka koje su najkorištenije, a to su: ExpressJS, Fastify, NestJS.

ExpressJS

ExpressJS je najstariji od svih spomenutih backend tehnologija, to je “brzi, minimalni razvojni okvir za NodeJS” [48]. Ima veliku zajednicu koja ga koristi koja sudjeluje u nadogradnji. Sukladno tome dobro je dokumentiran i postoje pregršt biblioteka koje su namijenjene za korištenje uz ExpressJS i pokrivaju gotove sve probleme sa kojima se možemo susresti. Ipak, nije idealno rješenje. Iako može pokriti gotovo sve slučajeve, brzina je manja od konkurenata, a originalni tim koji je razvio ExpressJS sada razvija drugu tehnologiju, Fastify.

Fastify

Fastify je “efikasni server koji ima nisku cijenu infrastrukture, bolju responzivnost pod stresom i sretne korisnike” [49]. Fastify je noviji okvir od ExpressJSa, samim time malo je manje popularan i ima manji sustav priključaka. Ipak, kako je originalni tim ExpressJSa sada odgovoran za razvoj Fastifya, zasigurno će postati novi standard. U odnosu na svog prethodnika znatno je brži, efikasniji i može posluživati više korisnika odjednom.

NestJS

Posljednji razvojni okvir u kontekstu backend tehnologija jest NestJS koji se definira kao “progresivni NodeJS razvojni okvir za izradu efikasnih, pouzdanih i skalabilnih serverskih aplikacija” [50]. NestJS svojom se brzinom nalazi između prethodnih razvojnih okvira, brži je od ExpressJSa, ali nešto sporiji od Fastifyja. Ipak, od svih spomenutih razvojnih okvira najviše je pridržan MVC arhitekturi i najboljim praksama iz domene serverskih aplikacija. Kako NestJS detaljno propisuje kako se mora pristupiti određenom problemu, kako se moraju definirati primjerice rute APIja, potrebno je duže vrijeme za razvoj nečega što se u drugim okvirima može kreirati sa svega 5 linija koda. Pozitivna strana ovog pristupa je pouzdanost. Kako se prate najbolje prakse, NestJS može činiti optimizacije nad kodom, a sam način pisanja koda smanjuje mogućnost greške programera. Uzimajući u obzir sve prednosti i nedostatke NestJSa, smatra se da nije uvijek idealan izbor. Iako je najodrživiji od svih spomenutih okvira,

problematika koju rješava ne bi trebala postojati u NodeJS svijetu. U slučaju da serverske aplikacije postaju toliko opsežne i zahtijevaju bolje performanse, bolje je odlučiti se za neke performantnije tehnologije.

Iako postoje mnogi drugi razvojni okviri za backend u JavaScript ekosustavu, spomenuti ExpressJS, Fastify i NestJS nakorišteniji su i u pravilu ne postoji razlog za odmicanje od njih. Također, u jednom od narednih poglavlja biti će objašnjeni i tzv. “fullstack” razvojni okviri koji već imaju integrirane neke od ovih backend tehnologija. U tom slučaju, ovaj odabir nije potreban, osim u slučaju kada aplikacija zahtijeva specifične implementacije izvan mogućnosti tih programskih okvira.

6.3.2. Frontend tehnologije

Suprotno backendu, pod pojmom frontend smatra se “sav kod koji se izvršava na web pregledniku korisnika” [2]. Najjednostavniji primjer tehnologija uključenih u ovu domenu jest kombinacija HTML, CSS i JavaSript, koji čine temelj većine aplikacije na webu. Ovaj minimalni odabir tehnologija omogućuje izradu svake web aplikacije, no ovako minimalan odabir sadrži i negativnu stranu. Kreiranje svega ispočetka veoma je zamorno i sporo naspram svemu što danas postoji u domeni web razvoja. Iz tog razloga stvorene su razne tehnologije čiji je cilj pojednostaviti i ubrzati razvoj.

Trenutno stanje frontend tehnologija je veoma burno. Postoje desetci razvojni okvira za JavaScript, tehnologije temeljene na predlošcima i sl. Cilj ovog poglavlja je usporediti neke od tih tehnologija koje omogućavaju najbrži i najkvalitetniji razvoj web aplikacija s TypeScriptom. Prema anketi StackOverfowa za 2024. godinu, neke od najaktualnijih i najkorištenijh tehnologija danas su: React, Svelte, SolidJS, Astro i Vue [51]. Valja napomenuti kako svaka od tih tehnologija ima prednosti i nedostatke. Iako je moguće implementirati jednake funkcionalnosti u svima njima, način, brzina i kvaliteta implementacije ovise o problemu. U nastavku će spomenute tehnologije biti bolje definirane te će se usporediti slučajevi korištenja istih.

React

React je “biblitoteka za web i nativna korisnička sučelja” [34] koja je izrađena i održavana od strane poduzeća Meta. Već duži niz godina je najpopularnija JavaScript biblioteka što je odraženo u svim analizama JavaScript biblioteka, velikom broju programera koji odabiru React kao početnu točku te broju aplikacija razvijenih u Reactu. React je odličan za visoko interaktivna sučelja, ali zbog virtualizacije elemenata i načina na koji uspoređuje trenutno stanje stranice s budućim ima smanjene performanse pri prikazu velikih količina podataka te

zahtijeva dobro poznavanje tehnologije kako bi se izradile kvalitetne aplikacije. Također, React sadrži pregršt funkcionalnosti te postoji velika zajednica programera koji su razvili napredne biblioteke za mnoštvo raznih slučajeva.

Svelte

Svelte je jedan od najnovijih JavaScript biblioteka za razvoj performantnih web aplikacija. Ova biblioteka razvija se kao projekt otvorenog koda, ali je podržana od strane poduzeća Vercel. Glavna svrha Svelta je optimizacija koda pri izgradnji aplikacije kako bi osigurala minimalan i optimalan kod koji se mora izvršiti u samom web pregledniku. Iako se uglavnom koriste samo osnovni koncepti JavaScripta, Svelte posjeduje specifičnu implementaciju određenih značajki i zahtjeva učenje istih. Primjerice, reaktivnost stranice postiže se normalnom deklaracijom varijable kao i u JavaScriptu, i njenim pozivanjem unutar HTML predloška [35].

SolidJS

Razvijen je s namjerom implementacije najboljih aspekata Reacta, ali bez problema s performansama, SolidJS se prema službenoj dokumentaciji "osjeća prirodno jer slijedi istu filozofiju kao React" [36]. Najveća razlika, i razlog razlici performansi, jest korištenje signala umjesto virtualizacije. Slično kao u slučaju Reacta, SolidJS posjeduje specifičnosti u pisanju koda i implementaciji te zahtijeva dobro poznavanje tehnologije kako bi aplikacije bile funkcionalne i kvalitetne.

Vue

Još jedan od veoma popularnih razvojnih okvira za frontend jest Vue. Pisanje programskog koda sa Vueom je relativno slično pisanju čisto HTML i JavaScripta, sa dodatnim značajkama. Slično kao dosad spomenuti razvojni okviri omogućuje izradu interaktivnih slučajeva. Programeri koji su razvili Vue odlučili su smanjiti detaljne mogućnosti React kako bi postigli minimalniji razvojni okvir sa manjom količinom podataka koje je potrebno preuzeti kako bi mogao raditi [38].

Astro

Za razliku od ostalih frontend biblioteka koje su fokusirane na dinamičke aplikacije, Astro je prvenstveno namijenjen izradi statičnih stranica sa djelomično dinamičkim komponentama. Primjer jedne takve stranice, bila bi statična web stranica za blog koja ima potrebu za dinamičkim gumbom za sviđanje određene objave [37].

Radi bolje preglednosti i posebnih slučajeva korištenja spomenutih razvojnih okvira u narednoj tablici uspoređene su glavne točke za svaki. U prvom stupcu, “Biblioteka” nalazi se naziv biblioteke, u naredna dva stupca raspisane su prednosti i nedostaci naspram drugima, dok se u posljednjem stupcu nalaze određeni slučajevi kada je idealno koristiti tu biblioteku.

Tablica 2. Usporedba frontend biblioteka

Biblioteka	**Prednosti**	**Nedostaci**	**Slučajevi korištenja**
React	- velika zajednica i broj biblioteka - pokriva najveći broj potencijalnih zahtjeva aplikacije	- problemi sa performansama pri prikazu velike količine sadržaja - potrebno duboko znanje za kvalitetnu implementaciju	- dinamičke i izrazito interkativne web aplikacije
Svelte	- efikasan i minimalan u web pregledniku - uključeno rješenje za varijable (eng. *state*) - uglavnom koristi osnovne koncepte JavaScripta - perfromantan	- relativno mala zajednica	- dinamičke web aplikacije - minimalne i efikasne aplikacije
SolidJS	- najperformatnija biblioteka	- relativno nova biblioteka koja je još uvijek u razvoju i nema sve pogodnosti drugih biblioteka	- dinamičke web aplikacije
Astro	- performantnost - moguće je koristiti bilo koju drugu od spomenutih razvojnih okvira uz Astro	- nije namijenjeno za izrazito dinamičke web aplikacije	- djelomično dinamičke ali generalno statične stranice
Vue	- dokumentacija i ostali materijali prevedeni u mnoštvo jezika - mala veličina biblioteke	- relativno mala zajednica	- dinamičke web aplikacije

Kao što se može zaključiti iz prethodne tablice, slučajevi korištenja razvojnih okvira uglavnom se preklapaju, stoga nije nužno pretjerano razmatrati koju od spomenutih tehnologija koristiti. One su bile odabrane prema popularnosti i kvalitetnoj integraciji sa ostalim tehnologijama i servisima spomenutima u ovome radu te TypeScriptom, iako postoje drugi kao što je Remix [42]. Izuzev potrebe za određenim funkcionalnostima, za implementaciju korisničkog sučelja u pravilu se može koristiti bilo koja tehnologija sa kojom je programer upoznat.

U nastavku će biti objašnjene “full-stack” tehnologije pri čemu će neke od dosad spomenutih frontend i backend tehnologija biti spojeni u jedno. Ti razvojni okviri nude širi spektar mogućnosti nego same frontend tehnologije, no ponekad nisu potrebne sve funkcionalnosti koje nude.

6.3.3. Fullstack tehnologije

Do sada su bile razmatrane frontend i backend tehnologije zasebno, pri čemu frontend karakterizira kod koji će se izvršavati na strani web preglednika, dok se backend izvršava na serveru. Ipak, u većini slučajeva za ispravan rad web aplikacije potrebne su obje komponente, frontend i backend. Radi toga, određeni programeri uvidjeli su potrebu za boljom integracijom između te dvije strane. Rezultat toga jesu fullstack tehnologije, odnosno tehnologije koje spajaju značajke frontend i backend tehnologija. U današnjem JavaScript svijetu, fullstack tehnologije imaju velik opseg korištenja i stvoren je velik broj takvih razvojnih okvira, svaki od kojih rješava neke probleme, ali i donosi svoje određene probleme. Neki od trenutno najkorištenijih i najpopularnijih fullstack razvojnih okvira su NextJS, Nuxt, SvelteKit, SolidStart i Remix. U nastavku će biti objašnjen svaki od njih, njihove specifičnosti i mane te kada odabrati koji ovisno o zahtjevima aplikacije.

NextJS

NextJS pripada u kategoriju meta razvojnih okvira (eng. *meta framework*), odnosno razvojni okvir za razvojni okvir React. Stvoren je i održavan od poduzeća Vercel i omogućuje korištenje serverskih mogućnosti u sklopu sa Reactom. NextJS trenutno je jedan od najpopularnijih razvojnih okvira za JavaScript i pruža najveći skup najnovijih mogućnosti Reacta. Omogućuje kreiranje kako statičnih, tako i dinamičkih web aplikacija te je preporučen način za pokretanje novih projekata od samog React razvojnog tima. Jednostavna datotečna struktura stranica, pri čemu svaka putanja web stranice odgovara datoteci u definiranoj mapi, pojednostavljuje i ubrzava razvoj, dok ugrađeni API server eliminira potrebu za dodatnim postavljanjem sličnog takvog servisa [39].

Nuxt

Slično kao NextJS, Nuxt je meta razvojni okvir, ali za Vue.js koji je fokusiran na razvoj serverskih aplikacija, jednostraničnih aplikacija, kao i statičnih stranica. Također uključuje datotečno postavljanje ruta, generiranje statičnih datoteka te razne optimizacije za samu web stranicu [43].

SvelteKit

SvelteKit je službeni razvojni okvir za razvijanje aplikacija sa Svelte jezikom. Uključuje sve pogodnosti kao NextJS i Vue.js, no razlikuje se u sintaksi i samom prevoditelju koda, koji odrađuje razne optimizacije pri izgradnji produkcijske verzije gotove aplikacije kako bi stvorio minimalan produkcijski kod [40].

SolidStart

Nastavljajući trend spomenutih razvojnih okvira, SolidStart je službeni razvojni okvir za istoimeni jezik, SolidJS. Iako omogućava sve pogodnosti kao i drugi razvojni okviri koji su bili spominjani, razlikuje se po načinu na koji odrađuje web zahtjeve, što u određenim slučajevima može poboljšati performanse aplikacije [41].

Povijesno, frontend i backend su bili potpuno odvojeni i programeri su se najčešće specijalizirali za jedno ili drugo te razvijali specifična znanja za to područje. Korištenjem novijih tehnologijama i razvojnih okvira, posebice u svijetu JavaScripta, granica između frontenda i backenda postaje sve zamućenija. Jedan od razloga jesu upravo fullstack razvojni okviri koji predstavljaju spoj frontend i backend tehnologija. Samim time, fullstack programeri postaju sve češći i traženiji. Iako odvajanje u specifično područje omogućuje programerima dublje znanje tog područja i popratnih tehnologija, neosporiva je činjenica da su fullstack programeri produktivniji i brži u razvoju funkcionalnih zahtjeva aplikacija. Mogućnost odrađivanja svih domena problema, uveliko pojednostavljuje razvoj aplikacije. Ipak, samim spajanjem domena, programeri moraju znati dovoljno o svakoj od njih što može predstavljati dodatnu prepreku. Više o najbitnijim temama o kojima je potrebno biti informiran pri razvoju fullstack aplikacija biti će razrađeno u nastavku.

6.3.4. Baza podataka

Jedan od najbitnijih aspekata svake aplikacije, ako je primjenjiv, jesu podaci. Sukladno tome, može se zaključiti kako je odabir tehnologije u kojoj će podaci biti spremani jednako bitan i znatno mijenja način na koji će aplikacija biti implementirana. U pogledu odabira baze podataka postoje mnoge značajke na koje valja pripaziti. Kako je područje baza podataka opsežno i promjenjivo ovisno o zahtjevima aplikacije, gotovo je nemoguće pokriti sve slučajeve. Stoga će u ovome poglavlju biti definirane neke od mogućnosti koje pokrivaju najveći broj potencijalnih zahtjeva aplikacije. Također valja napomenuti kako će naglasak biti na tehnologijama temeljenih na otvorenim standardima, kako one pružaju veću slobodu u implementaciji i eventualnim migracijama na druge tehnologije i/ili servise. Nadalje, biti će definirani i načini posluživanja baza podataka, odnosno servisi koji to omogućuju.

Prvo pitanje na koje je potrebno odgovoriti jest kakvog će oblika biti podaci, hoće li biti strukturirani ili nestrukturirani. Od relacijskih standardnih tehnologija valja spomenuti MySQL i PostgreSQL, a od baze podataka temeljene na dokumentima, primjerice MongoDB ili pak CouchDB. Kako bi se na temu odabira prikladne baze podataka mogao napisati zaseban diplomski rad bile su spomenute samo 4 vrste baza podataka, a u nastavku će fokus biti na način posluživanja baze podataka.

Kada se razmatraju načini posluživanja baze podataka, možemo izdvojiti dvije glavne podjele: samoposluživanje i servisi. Samoposluživanje baze podataka podrazumijeva posjedovanje servera na kojem će se instalirati i pokrenuti baza podataka te neki način na koji će se klijenti, odnosno aplikacije, moći povezati sa bazom podataka. Sa druge strane, servisi obuhvaćaju već gotove baze podataka koje se mogu zakupiti i jednostavno koristiti, no detaljnije će servisi biti objašnjeni u narednom poglavlju. Svaki od ova dva spomenuta načina posluživanja baze podataka imaju prednosti i nedostatke. Iako se samoposluživanje uglavnom pokazalo kao jeftinije mora se samostalno instalirati i održavati, što zahtijeva vrijeme od strane programera. Također, ako u danom trenutku baza podataka ima toliko prometa da server to ne može podnijeti, potrebno je skalirati server, što je sasvim novi skup problema. Suprotno tome, baze podataka kao servisi su od samog početaka skuplji, no u većini slučajeva može se u potpunosti zaboraviti na sve popratne probleme, kao skaliranje, jer njih rješava vlasnik tog servisa. Samim time prednost baze kao servisa jest da programeru jedino preostaje povezivanje na bazu i korištenje spomenute baze podataka. Neki od servisa za baze podataka koje valja izdvojiti prema funkcionalnostima, modernosti i korištenosti jesu: Supabase, PlanetScale, AWS Relational database i Turso.

6.3.5. Servisi

Programi kao servisi (eng. *Software as a Service*) ili skraćeno SaaS, omogućuju programerima povezivanje na aplikacije u oblaku putem interneta. Svi takvi servisi imaju neki od oblika plaćanja prema korištenosti (eng. *pay-as-you-go*) što na većoj skali predstavlja veće troškove nego vlastita implementacija, no znatno pojednostavljuju implementaciju.

U poglavlju 5.3.4. već je bilo spomenuto korištenje otvorenih standarda, prije nego što se definiraju potencijalni servisi, valja spomenuti dva bitna pojma u tom pogledu i razliku među njima, zaključavanje u poslužitelja (eng. *vendor lock-in*) i ugrađenost u poslužitelja (eng. *vendor built-in*). Prema Browneu (2023), zaključavanje u poslužitelja karakteriziraju sve odluke pri dizajniranju servisa od strane poslužitelja, koje otežavaju odmicanje od nj, dok ugrađenost u poslužitelj karakteriziraju sve dodatne mogućnosti pojedinog servisa koje pojednostavljuju rad korisnika tog servisa. Jedan primjer za zaključavanje u poslužitelja jest baza podataka u Firebaseu, Googleovoj platformi za razvoj aplikacija. Iako spomenuta baza podataka rješava

probleme poput skalabilnosti, autentifikacije, autorizacije i podataka u realnom vremenu, potrebno je koristiti posebne tipove podataka i strukturu. Ovaj standard definiran je od Googlea radi rješavanja spomenutih problema, no kako ga koristi jedino Firebase, ako se u bilo kojem trenutku želi odmaknuti od ove platforme potrebno je kreirati komplekse skripte za migraciju podataka. Naspram tome, za ugrađenost u poslužitelja dani primjer može biti već spomenuti Vercel, konkretno, njihova platforma za posluživanje aplikacija. Jedna od dodatnih značajki Vercelove platforme jest korištenje tzv. lambda funkcija, gdje se za svaki poziv API krajnje točke pokreće nova pojednostavljena instanca virtualnog privatnog servera koja omogućava gotovo neograničenu skalabilnost sustava [21]. Ova značajka ne zahtijeva nikakvu posebnu implementaciju od strane programera već radi čim se aplikacija poslužuje na platformi. Ako bi se ta ista značajka željela replicirati na vlastitom sustavu zahtijevala bi veliki napor programera u dizajnu i implementacije infrastrukture, no sam kod ne bi se morao nimalo mijenjati. U ovom primjeru sada se može zamijetiti prethodna karakteristika ugrađenosti u poslužitelja, to su funkcionalnosti koje “nisu neophodne već one koje pojednostavljuju rad korisnika tog servisa” [17].

Postoje mnogobrojni servisi za gotovo sve programske probleme na koje je moguće naići tokom razvoja aplikacija. Samim time obujam svih raznih servisa prevelik je za nabrajanje, no bitno je biti upoznat sa raznim servisima koji postoje te znati kada ih se potencijalno može iskoristiti za pojednostavljivanje razvoja web aplikacija.

6.3.6. Verzioniranje koda

Od početaka programiranja u timovima postojao je problem sinkronizacije koda između članova. Neovisno da li neka osoba dodaje, briše ili uređuje datoteku, na koncu svi uključeni članovi trebali bi imati jednake datoteke. Slijedno tome, jedan od najvećih napredaka u efikasnosti programiranja bilo je verzioniranje koda (eng. *code versioning*) ilitiga upravljanje izvornim kodom. Ono je “sustav koji zabilježava promjene na određenoj datoteci ili skupu datoteka tako da je moguće nakon određenog vremena vratiti prošlu verziju” [5]. Iako su postojali slični, no drugačiji načini verzioniranja u povijesti, nijedan nije bio kvalitetan kao trenutno najkorišteniji sustav za distribuirano verzioniranje koda, Git. Ključna riječ je distribuirani sustav, što u osnovi podrazumijeva postojanje centralnog servera na kojem se spremaju sve promjene, kao i lokalni server, koji predstavlja računalo programera. Takvim pristupom svaki se programer jednostavno može sinkronizirati sa glavnim serverom, neovisno o tome da li to znači preuzimanje najnovije verzije koda ili pak ažuriranje trenutne verzije.

Verzioniranje koda danas se podrazumijeva kada se radi na bilo kakvom programskom projektu, kao takvo veoma je bitno spomenuti koji sve postoje i argumentirati svaki od njih.

Kako je bilo spomenuto, danas najkorišteniji sustav je Git, no postoji nekolicina hvalevrijednih platformi i/ili tehnologija koje su razvijene na samom Gitu, a to su Github, Gitlab i BitBucket.

Github

Github je „vodeća platforma za verzioniranje koda sa više od 100 milijuna aktivnih korisnika, 4 milijuna organizacija i više od 420 milijuna aktivnih repozitorija“ [6]. Github je najveća takva platforma te nudi pregršt pogodnosti, ali ne rješava sve probleme idealno. U daljem tekstu biti će istaknuti pozitivni i negativni aspekti Github platforme.

Tablica 4. Prednosti i nedostaci Githuba za verzioniranje koda

Prednosti
- velika zajednica programera - ugrađeni CI/CD proces - pretraživanje koda - verzioniranje i moćni načini integracije koda - besplatne stranice za posluživanje jednostavnijih stranica - mogućnost kreiranja i praćenja zadataka projekta - skalabilan sustav - velik sustav priključaka
Nedostaci
- većina funkcionalnosti nije besplatna za privatne projekte - relativno visoke cijene ovisno o zahtjevima aplikacije - potencijalno zaključavanje u Githubov sustav

Github predstavlja polaznu točku za većinu projekata danas i sa svime spomenutim također se smatra da je za verzioniranje koda, osim u određenim situacijama, najbolje odabrati Github. Kako je najpoznatija platforma nudi integraciju sa većinom postojećih servisa koji se kasnije mogu koristiti. Pouzdan je i siguran način verzioniranja koda koji nije potrebno samostalno održavati, a velika tržnica priključaka rješava gotovo sve probleme na koje se potencijalno može naići u procesu razvoja verzioniranja koda i integracije u razvoj aplikacije. Jedina negativna strana jest potencijalna cijena, no u usporedbi sa drugim servisima na koncu je jeftinija ili neznatno skuplja za sve pogodnosti koje nudi.

Gitlab

Gitlab možemo karakterizirati kao „Github otvorenog koda“ [3]. Postoje određene razlike u implementaciji i funkcionalnostima za velika poduzeća, no u osnovi su identični. Najveća razlika jest ta da se Gitlab može samostalno posluživati na vlastitom serveru. Ovim pristupom Gitlab je postao prvi odabir za mnoge programere, ponajviše poduzeća koja bi u suprotnome plaćali velike iznose Githubu ili koji pak žele samostalno upravljati sigurnošću svoga koda. Ipak, takav pristup nije savršen, stoga će u narednom dijelu biti istražene njegove prednosti i nedostatci.

Tablica 5. Prednosti i nedostaci Gitlaba za verzioniranje koda

Prednosti
- ugrađen CI/CD proces - ugrađen registar kontejnera za Docker - skalabilan sustav - mogućnost samoposluživanja na vlastitom serveru - sve funkcionalnosti su besplatne
Nedostaci
- za manje projekte, ili jedan projekt, zahtijeva puno postavljanja - zahtijeva podosta računalnih resursa - potrebno je vlastito implementiranje i održavanje - manji sustav priključaka naspram konkurenciji

Gitlab kao proizvod poprilično je kvalitetan i dobra konkurencija Githubu. Ipak, smatra se da zahtijeva previše rada pri postavljanju i održavanju, ako mu je namjena samo za jedan projekt. U tom slučaju bolje je koristiti već uređenu Githubovu platformu, ako pak već postoji implementirani Gitlab sustav i/ili je Gitlab već od prije korišten za druge projekte, tada ne postoji razlog zašto ga ne koristiti.

BitBucket

Slično Githubu, BitBucket je platforma u oblaku izrađena da prati sve potrebe modernog web razvoja, što uključuje ali nije ograničeno na verzioniranje koda, CI/CD proces, praćenje projekta i vođenje dokumentacije. Kao i sa prethodnim primjerima, razmotriti će se prednosti i nedostaci ove platforme.

Tablica 6. Prednosti i nedostaci BitBucketa za verzioniranje koda

Prednosti
- integracija sa Jira sustavom za praćenje projekata - ugrađen CI/CD proces - besplatno za manje timove - velik sustav priključaka
Nedostaci
- zatvoren sustav priključaka - visoki troškovi - integracija sa drugim servisima (Jira) zahtjeva posjedovanje njihovih licenci - potencijalni problemi sa performansama

Od svih spomenutih sustava za verzioniranje koda BitBucket se ističe brojem funkcionalnosti, ali većina njih, nije potrebna i/ili postoje bolje alternative. Iako je također kvalitetno rješenje, BitBucket usko veže korisnika uz njihovu platformu, što može otežati migraciju prema drugim rješenjima kada je to potrebno. Ako se tek odlučuje koju od tehnologija za verzioniranje koda odabrati, ne preporuča se BitBucket, ali kao i sa GitLabom, ako je već korišten od strane korisnika ili poduzeća, tada predstavlja solidno rješenje.

Postoje još mnoge zanimljivosti i najbolje prakse tokom verzioniranja koda, kao i rasprave kako je najbolje verzionirati kod, kako dodavati nove funkcionalnosti i sl. Ipak, objašnjavanje svega toga izlazi van domene ovog rada, čija je svrha čisto odabrati najbolje tehnologije i servise.

6.4. Posluživanje aplikacije

Jedan od ključnih koraka u procesu razvoja aplikacija je odabir mjesta posluživanja aplikacije. Posluživanje aplikacije u ovom kontekstu znači “omogućavanje pristupa web aplikaciji sa uređaja korisnika“ [7]. U ovoj naizgled jednostavnoj definiciji krije se mnoštvo detalja koje je potrebno razraditi. Kako je već bilo definirano u poglavljima o Web aplikacijama i Arhitekturi web aplikacija, potreban je server ili servis koji će posluživati web aplikaciju prema svim korisnicima. Već na samom početku postoje dva potpuno različita pristupa, a to su samostalno posluživanje (eng. *self-hosting*) ili servisi za posluživanje (eng. *web hosting services*). Kako im imena dadu naznaku, pri samostalnom posluživanju korisnik upravlja cijelim serverom, ili dijelom servera, dok servisi za posluživanje uglavnom zahtijevaju samo datoteke

koje su potrebne za pokretanje aplikacije uz minimalnu konfiguraciju, kako bi aplikacija bila poslužena. Nešto detaljnija razdioba kategorizira web posluživanja u 6 generaliziranih vrsta, a to su [8]:

- dijeljeno posluživanje (eng. *shared hosting*)
- posluživanje na virtualnom privatnom serveru (eng. *VPS hosting*)
- dedicirano posluživanje (eng. *dedicated hosting*)
- upravljano posluživanje (eng. *managed hosting*)
- posluživanje u oblaku (eng. *cloud hosting*)
- kolokacijsko posluživanje (eng. *colocation*)

Dijeljeno posluživanje

Dijeljeno posluživanje najjeftinije je ali i najograničenije posluživanje od svih spomenutih. U takvoj vrsti posluživanja, korisnik dobiva pristup dijelu servera koji je dijeljen sa brojnim drugim korisnicima. Iako ovakav pristup može biti financijski isplativiji, programeru otežava posao jer ne može jednostavno povećavati zakupljene resurse koji mogu biti potrebni za posluživanje web aplikacije. Također, velik dio ovakvih servisa ograničava količinu računalnih resursa koje je moguće zakupiti, tako će se, ako broj korisnika dovoljno poraste, svejedno morati napustiti dijeljene servise. Nadalje, ovakvi servisi često su ograničeni tehnologijama koje se mogu koristiti i u pravilu pokrivaju samo najosnovnije. Iako postoje izuzeci, servisi za dijeljeno posluživanje generalno mogu posluživati samo statične HTML, CSS i JavaScript datoteke, PHP ili pak jednostavne NodeJS servere.

U narednoj tablici uspoređene su cijene i mogućnosti 3 najpoznatija dijeljena servisa NameCheap, Hostinger i BlueHost. Za svaku od servisa uspoređene su mogućnosti sa sličnim cijenama kako bi se bolje mogao prikazati spektar mogućnosti koje nude.

Tablica 7. Usporedba servisa za dijeljeno posluživanje (Izvor: Vlastita izrada prema [9], [10], [11])

Naziv	Cijena	Pogodnosti
NameCheap (Stellar Business)	9.48$ / mjesečno	- besplatna domena - neograničen broj web stranica - 50GB pohrane - neograničen broj email pretinaca - interaktivni razvoj stranice - automatski povrat podataka i pohrana u oblaku
Hostinger (Business)	8.99$ / mjesečno	- 100 web stranica - 200GB pohrane - dnevno spremanje i mogućnost povrata

		podataka - neograničen promet - besplatna domena i email - interaktivni razvoj stranice - sigurnosni alati
BlueHost (Basic)	11.99$ / mjesečno	- domena (1. god) - CDN - 10GB pohrane - cca. 15.000 posjetitelja mjesečno - praćenje zaraženih datoteka

Promatrajući rezultate iz tablice 7, može se uočiti kako najviše pogodnosti za najmanju cijenu nudi Hostinger u “Business” opciji. Moguće je kreirati najviše stranica, dobiveno je 4 puta više prostora za pohranu nego pri NameCheapu i 20 puta više nego pri BlueHostu te najbitnije, nema ograničenja na promet stranice.

Kako je bilo uočljivo iz opisa o dijeljenim servisima za posluživanje, veoma su ograničeni i kao takvi mogu služiti samo za najosnovnije, statične aplikacije i aplikacije sa malim prometom. Ako su zahtjevi aplikacije unutar ovih ograničenja tada je dijeljeno posluživanje najisplativija opcija, ponajviše jer velik dio ovih servisa ima odlične alate za upravljanje aplikacijama, kao što je bilo vidljivo u usporedbi u tablici 7. Ipak, kako većina aplikacija izlazi iz spomenutog okvira ograničenja uglavnom se ne preporučuju dijeljeni poslužitelji.

Posluživanje na virtualnom privatnom serveru

Kako bi se moglo definirati posluživanje na virtualnom privatnom serveru, VPS u nastavku, prvo je potrebno definirati što VPS jest. „Ono je virtualni i izolirani dio fizičkog servera kojim upravlja poslužitelj, što se postiže tehnologijama za virtualizaciju“ [12]. Slično kao sa dijeljenim posluživanje i dalje se server dijeli sa drugim korisnicima, no ovdje svaki korisnik u potpunosti upravlja vlastitim dijelom. Pojednostavljeno, svaku od VPS instancu možemo promatrati kao vlastito računalo na koje je moguće instalirati što god programer želi, izuzev operacijskog sustava. Iako ovim pristupom programer ima veću kontrolu nad serverom, istovremeno je zadužen za konfiguraciju samog servera kako bi mogao raditi. Primjerice, postavljanje vatrozida, postavljanje balansera opterećenja, instalacija potrebnih biblioteka i sl.

U narednoj tablici uspoređena su 3 najpoznatija i najpouzdanija VPS servisa, DigitalOcean, Amazon Web Services (skraćeno AWS) i Fly. Bitno je napomenuti kako su uspoređene razine u sličnom rangu cijena, iako svaki od servisa ima opcije koje pokrivaju i najzahtjevnije aplikacije no koštaju više tisuća dolara po VPS instanci.

Tablica 8. Usporedba servisa za VPS posluživanje (Izvor: Vlastita izrada prema [13], [14], [15])

Naziv	Cijena	Pogodnosti
DigitalOcean (Basic Droplet)	32$ / mjesečno	- 4GB radna memorija - 2 virtualna procesora - 4TB izlaznog prometa (nakon 0.01$/GB) - 120GB pohrane
AWS (EC2 on demand)	24$ / mjesečno	- 4GB radna memorija - 2 virtualna procesora - 100GB izlaznog prometa (nakon 0.09/GB)
Fly.io (Launch)	22.65$ / mjesečno	- 4GB radna memorija - 2 virtualna dijeljena procesora - 100GB izlaznog prometa (nakon 0.02$/GB)

Valja napomenuti kako sve ove cijene variraju ovisno o prometu i stvarnoj potrošnji. Primjerice, iako je AWS nešto jeftiniji za iste računalne resurse kao DigitalOcean, imaju 40 puta manje besplatnog prometa i 9 puta višu cijenu po GB nakon besplatnog ograničenja. Slično tome, AWS i Fly imaju jednak izlazni promet no Fly-jev je jeftiniji za 0.07$ po GB. Ako se u obzir uzme da web aplikacija sa srednjim brojem korisnika ima promet od 10TB to je razlika od 700$. Također, svaki od servisa zna imati neke “sakrivene” cijene radi kojih je potrebno vrlo dobro proučiti sve uvjete korištenja, kako račun ne bi ispao veći od očekivanoga. Nadalje, jedan od problema kod svih ovih servisa je skalabilnost. Prvo je potrebno implementirati vlastite način praćenja sustava kako bi se znalo kada je potrebno zakupiti više računalnih resursa, odnosno kada ih se može raspustiti, kako se ne bi nepotrebno trošila novčana sredstva na njih, a zatim i implementirati sam način zauzimanja i oslobađanja. O ovoj problematici moguće je napisati zaseban rad samo na činjenici koliko mnogo raznih rješenja postoji, no za potrebe ovog rada dovoljno je spomenuti kako svaki od servisa nudi funkcionalnosti koje rješavaju tu problematiku na njihovoj platformi, ali po određenoj cijeni. Tako svaka aplikacija može biti poslužena milijunima korisnika bez pretjerane brige sa strane programera.

Sa svime spomenutim, VPS-ovi se smatraju kao odlična opcija ako postoje visoki ili specifični zahtjevi što određena aplikacija mora činiti. Iako zahtijevaju više rada i znanja, osim uzdržavanja vlastitih fizičkih servera gotovo ne postoji svestranija mogućnost nego VPS posluživanje. Ipak, sve ove pogodnosti imaju novčanu cijenu, kako direktno za servis, tako i za programera koji će to morati održavati. Generalno, ako aplikacija ima zahtjeve koji nisu unutar okvira drugih rješenja za posluživanje, smatra se da je VPS idealan opcija za sve potrebe.

Dedicirano posluživanje

Dedicirano posluživanje slično je VPS posluživanju u smislu da je programer taj koji ima mogućnost upravljanja serverom. Ipak, dedicirani serveri idu korak dalje, pri čemu korisnik ne zakupljuje virtualni dio dijeljenog servera, već zakupljuje čitav fizički server od nekog poslužitelja. Programer može činiti sa tim server što god želi, počevši već od odabira operacijskog sustava za instalaciju. U sljedećoj tablici uspoređena su 3 servisa za dedicirano posluživanje koji se najčešće spominju u tom kontekstu, a to su Liquid Web, A2 Hosting i HostGator. Kao i u prethodnim primjerima, uzeti su u obzir servisi u prosječnom jednakom rangu snage servera po računalnim resursima, iako postoje i snažniji.

Tablica 9. Usporedba servisa za dedicirano posluživanje (Izvor: Vlastita izrada prema [18], [19], [20])

Naziv	Cijena	Pogodnosti
Liquid Web	99$ / mjesečno	- 16GB radna memorija - 4-ero jezgreni procesor, 3.4GHz - 480GB SSD pohrane - 5TB izlaznog prometa
A2 Hosting	155.99$ / mjesečno	- 16GB radna memorija - 4-ero jezgreni procesor, 4.6GHz - 2TB SSD pohrane - 6TB izlaznog prometa
HostGator	89.98$ / mjesečno	- 8GB radna memorija - 4-ero jezgreni procesor, 2.1GHz - 1TB HDD pohrane - neograničen izlazni promet

Serveri za dedicirano posluživanje dvostruko su, ili čak više, skuplji nego slični serveri pri VPS posluživanju. Samim time ovakvi serveri uglavnom su korišteni samo od strane poduzeća.

Upravljano posluživanje

Upravljano posluživanje najčešće se spominje u kontekstu servisa u kojima je moguće kreirati i poslužiti gotovu stranicu, primjerice WordPress. Upravljani poslužitelji rješavaju podosta problema koje bi programer drugačije morao sam rješavati i održavati. Najbolji primjeri takvih problema jesu konstanta optimizacija stranice za web pretraživače, ažuriranje sigurnosnih aspekata radi sprječavanja prodora, sigurnosne zakrpe i povremeno spremanje radi povrata podataka. Drugim riječima, upravljano posluživanje tehnički pripada dijeljenom posluživanju kao što je bilo opisano u poglavlju o Dijeljenom posluživanju. Također, danas se mnogi servisi

za dijeljeno i upravljano posluživanje preklapaju, pa tako svi primjeri iz tablice 7 i obrazloženje za njihovo korištenje ili izbjegavanje vrijede kako je objašnjeno u tom primjeru.

Posluživanje u oblaku

Posluživanje u obliku steklo je veliku popularnost u zadnjih nekoliko godina, često se uz nj veže naziv računalstvo bez poslužitelja (eng. *serverless computing*). Razlika posluživanja u oblaku naspram primjerice VPS posluživanja na oblaku jest ta da se ne zakupljuje nikakav zasebni server, već se plaća ovisno o iskorištenim resursima. Ako web aplikacija trenutno nema korisnika, potrošnja računalnih resursa biti će gotovo nula, ako dođe tisuću korisnika automatizirano će se zauzeti potrebni dio resursa, a ako dođe 10 milijuna korisnika, samo će se zauzeti još više resursa servera poslužitelja. Ovaj način posluživanja vrši se na infrastrukturi poslužitelja koja je često distribuirana diljem svijeta. Tako primjerice, korisnici na raznim dijelovima svijeta mogu primati podatke iz raznih podatkovnih centara poslužitelja koji su im najbliži. Samim time, posluživanje u oblaku nudi razne prednosti poput automatizirane skalabilnosti, visoke sigurnosti, najnovije računalne opreme i tehnologije. U kontekstu posluživanja u oblaku trenutno su aktualna dvije novije tehnologije, iako se nazivi razlikuju kod raznih servisa, često se nazivaju Edge i Lambda funkcije. U narednoj tablici uspoređena su 2 najpopularnija servisa za posluživanje u oblaku, Netlify i već spominjani Vercel.

Tablica 10. Usporedba servisa za posluživanje u oblaku (Izvor: Vlastita izrada prema [21], [22], [23])

Naziv	Cijena	Pogodnosti
Vercel	20$ / korisnik mjesečno	- 1 milijun poziva Edge funkcija (zatim 2$ / 1 milijun) - 1TB prometa (zatim 40$ / 100GB) - 1000 - 24.000 minuta izgradnje koda - neograničen broj stranica (domena) - 1000 GB-sati izvođenja API funkcija (zatim 40$ / 100 GB.sati)
Netlify	19$ / korisnik mjesečno	- 500 web stranica - 2 milijuna poziva Edge funkcija (zatim 2$ / 1 milijun) - 1TB prometa (zatim 55$ / 100GB) - 25.000 minuta izgradnje aplikacija

Pojašnjenje:

- Poziv Edge funkcije je sinoniman sa 1 API zahtjevom
- GB-sat računa se kao: (radna memorija zauzete instance u GB) * (sekunde izvršavanja) / 3600

Kolokacijsko posluživanje

Posljednja vrsta posluživanja je kolokacijsko posluživanje. Ova vrsta posluživanja nije primjenjiva za većinu korisnika, kako zahtijeva posjedovanje vlastitog fizičkog servera. Kolokacijski centri za posluživanje služe za čuvanje i uzdržavanje fizičkih servera koje im korisnici privremeno predaju. Zauzvrat, korisnik sa centrom dijeli dio troškova, ali najčešće ti troškovi budu manji nego da samostalno održava te iste servere. Ovaj pristup uglavnom koriste poduzeća koja imaju veliki broj fizičkih servera i žele smanjiti troškove održavanja centara podataka. Kao takav, ovaj pristup nije preporučen generalnim privatnim korisnicima.

6.5. Praćenje i bilježenje sustava

Svaka aplikacija u produkcijskoj okolini trebala bi imati sustav za nadziranje. Od osnovnih mogućnosti poput praćenja broja posjeta određenoj stranici, sve do detaljnjih zapisa o greškama koje se dešavaju tokom korištenja. Pregledni podaci o korištenju određenih funkcionalnosti nezaobilazne su stavke koje osiguravaju kvalitetnu, zdravu i performantnu web aplikaciju. U ovome kratkom uvodu bile su spomenute dvije kategorije nadzora sustava, praćenje (eng. *monitoring*) i bilježenje (eng. *logging*). Iako im se određene karakteristike podudaraju, zasebno će biti objašnjen svaki od njih kako bi se detaljnije moglo analizirati od čega se sastoje, kako ih ispravno implementirati te na što valja pripaziti. Glavna svrha praćenja sustava jest praćenje performansi i ponašanja pojedine aplikacije u realnom vremenu, što uključuje, ali nije ograničeno na metrike poput: zauzeća procesora, zauzeća radne memorije ili kašnjena u slanju zahtjeva (eng. *latency*). Za razliku od praćenja sustava, bilježenje sustava fokusirano je na praćenje pojedinih događaja unutar aplikacije, primjerice zahtjev prema bazi podataka, trenutci kreiranja stavki, zapisivanje pogrešaka i sl.

Na tržištu postoje brojni alati za praćenje i bilježenje sustava. Njihov odabir znatno ovisi o zahtjevima projekta, količini podataka koje je potrebno obraditi, jednostavnosti primjene ovisno o određenim tehnologijama itd. Metrike igraju veliku ulogu u praćenju performansi servera i otkrivanju potencijalnih problema u radu aplikacije. Servisi i alati koji će biti objašnjeni u nastavku odabrani su prema dugotrajnosti postojanja i kvaliteti usluge koju pružaju. Dodatno, odabrani su i neki od novijih servisa koji su trenutno aktualni na tržištu, ali nude pregršt pogodnosti. Pod ovime se ne podrazumijevaju biblioteke za logiranje kao što je Winston [30] ili Pino [31].

Prometheus i Grafana

Prvi od alata koji će biti objašnjeni jest Prometheus u kombinaciji sa Grafanom. Iako su to dva različita alata, dugi se niz godina koriste zajedno, pri čemu Prometheus prikuplja podatke sa servera, a Grafana omogućuje obrađivanje i vizualizaciju podataka. Oba alata su otvorenog koda, dobro testirani i korišteni od strane velikog broja poduzeća. Glavna značajka Prometheusa jest mogućnost prikupljanja metrika kao vremenske serije podataka, pri čemu se uz svaki zabilježeni podatak sprema i vrijeme u kojem je zapis kreiran [24]. Vrste podataka koje se mogu bilježiti su različite, a neki od mogućih podataka jesu: vremena obrade zahtjeva, broj aktivnih veza servera ili broj izvršenih zahtjeva. Grafana omogućuje preuzimanje podataka iz svih mogućih izvora. Unaprijed definiran dizajn sučelja, mnoštvo komponenata kao i brojni priključci pokrivaju gotovo sve slučajeve na koje se može naići u domeni vizualizacije podataka [25]. Iako je kombinacija Prometheusa i Grafane testirana i kvalitetna opcija, najveći nedostatak je potreba za vlastitim posluživanjem na serveru te sama implementacija.

Posthog

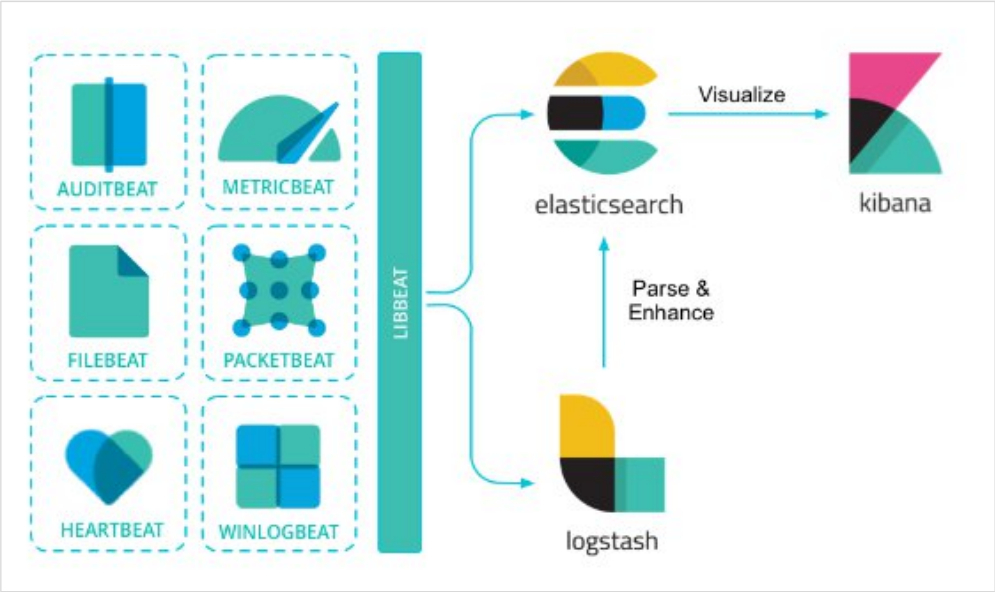
Posthog se predstavlja kao "jedinstvena platforma za analiziranje, testiranje, promatranje i posluživanje novih funkcionalnosti" koja predstavlja "jedinstvenu platformu za analitiku koja nativno radi sa reprodukcijom sesija, zastavicama za funkcionalnosti (eng. *feature flag*), A/B testiranje i anketama." [26] Izuzev brojnih funkcionalnosti koje ovaj servis nudi u oblaku, moguće ga je i samostalno posluživati na vlastitim serverima jednostavnim pokretanjem putem Dockera. U slučaju da se odabere mogućnost samoposluživanja, sve funkcionalnosti su besplatne. U suprotnom slučaju, ako se odluči koristiti infrastruktura kao servis, nude darežljivu besplatnu razinu i relativno jeftinu plaćenu razinu, u usporedbi sa konkurentima. Usporedba razina plaćanja, kao i pogodnosti koje su ponuđene, vidljive su u tablici 11.

Plausible

Web programeri su do nedavno uglavnom koristili Google Analytics za anonimnu web analitiku kao standard. Nakon što se iskustvo korištenja pogoršalo i nakon problema sa određenim pravnim zahtjevima od strane Europske Unije, koje Google Analitycs nije uspio uzdržati, javila se potreba za boljim alternativama. "Plausible je intuitivan, jednostavan sustav otvorenog koda za web analitiku" [27] koji je stvoren kako bi bio u skladu sa sigurnosnim standardima kao što su GDPR, CCPA i PECR. Korištenje ovog servisa veoma je jednostavno, potrebno je kreirati račun i uključiti danu skriptu u zaglavlje web aplikacije. Skripta će od tog trenutka anonimno pratiti sve najbitnije metrike stranice, kao što su: broj posjeta web aplikaciji, broj posjeta individualnim stranicama, trajanje posjeta stranici i sl. Slično kao i sa prethodno spomenutim servisom, Posthogom, postoji darežljiva besplatna razina, plaćena razina i mogućnost samoposluživanja.

ElasticStack

Elastic Stack, popularno zvan i ELK, sastoji se od 4 alata poduzeća Elastic, a to su: ElastiSearch, Kibana, Beats i Logstash. Ovaj skup alata omogućuje “pouzdano i sigurno preuzimanje podataka iz bilo kojeg izvora, bilo kojeg formata te pretraživanje, analizu i vizualizaciju istih.” [29] U cjelokupnom procesu, ElasticSearch je distribuirana, RESTful tražilica za pretragu podataka. Kibana omogućuje kreiranje analitike iz danih podataka sa fokusom na sklabilnost, preglednost, sigurnosti i mogućnost pretrage. Logstash, slično kao već spomenuta Grafana, je lanac za procesiranje podataka koji preuzima podatke iz višestrukih izvora, transformira ih i sprema na željenu lokaciju. Beats je spojnica između spomenutih alata i služi kao način prenošenja podataka iz Logstasha u Kibanu i ElastiSearch. Za bolje razumijevanje spoja alata, promotrimo sliku 2.



Slika 2. Vizualizacija ELK stack-a (Izvor: [ELK], preuzeto: 08.02.2024.)

.

Kao i svi dosad spomenuti alati, ELK je moguće koristiti kao plaćeni servis ili pak samostalno posluživati. Valja napomenuti kako je ELK korišten od strane velikih poduzeća što pridodaje njegovom kredibilitetu, ali je istovremeno i najskuplja od svih mogućnosti.

U tablici 11. prikazana je usporedna analiza svih spomenutih alata za praćenje i bilježenje sustava. Definirane su najbitnije karakteristike, cijena, prednosti i nedostaci. Samim time moguće je tokom odabira tih tehnologija odabrati najpogodniju za trenutne zahtjeve razvoja aplikacije.

Tablica 11. Usporedba alata za praćenje i bilježenje (Izvor: Vlastita izrada prema [25], [26], [27], [28], [29])

Naziv	Cijena	Prednosti	Nedostaci
Prometheus i Grafana	Besplatno	- otvoreni kod - mogućnost samoposluživanja - napredni filteri - unaprijed definirani ured dizajn sučelja i komponenata	- potrebno je posluživanje na vlastitom serveru - implementacija zahtjeva dublje znanje ovih alata - zahtjevnost implementacije sa određenim JavaScript razvojnim okvirima
Posthog	9€/mj. i više	- mogućnost samoposluživanja - jednostavna integracija - intuitivno sučelje za pregled i analizu podataka - mogućnost izrade detaljnih filtera i pregleda za sve postojeće podatke	- najjeftinija razina plaćenog servisa nema sve pogodnosti
Plausible	9€/mj. i više	- mogućnost samoposluživanja - jednostavna integracija - u skladu sa GDPR, CCPA i PECR direktivama - intuitivno sučelje za pregled i analizu podataka - slanje upozorenja preko emaila	- sadrži samo osnovne metrike za praćenje sustava
ELK	90€/mj. i više	- mogućnost samoposluživanja - najviše pogodnosti - testirano i korišteno od strane velikih poduzeća	- najskuplja od svih alternativa - u slučaju samoposluživanja najzahtjevnije u pogledu zauzeća procesora i radne memorije

Kao što se dade zaključiti iz tablice 11. svaki od objašnjenih alata i/ili servisa ima određene prednosti i nedostatke. Odabir idealne opcije ponajviše ovisi o mogućnosti programera, odnosno posjedovanju vlastitog servera, vremena koje je moguće utrošiti na implementaciju kao i maksimalne razine troškova. Kao što je bilo objašnjeno i kod većine drugih servisa, najbolja opcija u većini slučajeva predstavlja presjek alata koji je najjednostavniji za koristiti i najisplativiji. Prema tim karakteristikama, dva od četiri spomenuta alata se ističu, Posthog i Plausible. Jednostavnost implementacije kao i pregršt pogodnosti koje nudi, predstavljaju rješenje za većinu zahtjeva koji moraju biti zadovoljeni pri izradi web aplikacije. Pri tome Plausible omogućuje praćenje generalnih metrika sustava, dok Posthog nudi mogućnost detaljnog bilježenja svih akcija koje se izvode u sustavu.

6.6. Nadogradnja aplikacije

U današnjem dinamičkom razvoju aplikacija i konstantnoj težnji za izradom novih zahtjeva treba postojati jednostavan način kako dostaviti gotove funkcionalnosti. U proces nadogradnje aplikacije ubrajamo sve korake koje je potrebno izvesti kako bi se najnovija verzija koda mogla reflektirati u aplikaciju u produkcijskoj okolini, odnosno onoj aplikaciji koju korisnici u stvarnosti koriste. Iako naizgled jednostavan proces, u pozadini krije mnoge korake koje je potrebno izvesti kako bi se osigurala kvaliteta cijelog procesa. Samo neki od tih koraka su integracijsko testiranje, dobivanje povratnih informacija od programera i korisnika. U nastavku će biti detaljnije objašnjen proces nadogradnje aplikacije u sklopu sa postojećim alatima kreiranim u tu svrhu.

6.6.1. CI/CD

U svijetu brzog razvoja aplikacija programerima su potrebni prikladni alati koji će im omogućiti da nesmetano programiraju i rješavaju zahtjeve klijenata. Kontinuirana integracija i kontinuirana dostava (eng. *continuous integration and continuous delivery*), češće samo skraćeno CI/CD, je proces koji nastoji pojednostaviti i skratiti vrijeme između napisanog programskog koda i implementacije tog koda na stvarnoj aplikaciji. Drugim riječima, cilj je skratiti proces od trenutka kada se promjene u kodu reflektiraju u aplikaciji u stvarnom svijetu. U zadnjim godinama CI/CD je postao gotovo nezaobilazan dio razvoja svake aplikacije s dobrim razlogom. Promatrajući svaki od dva dijela CI/CD-ja, CI i CD zasebno, možemo uvidjeti čime konkretno pridonose cijelom procesu.

Sam CI je „praksa integracije koda u manjim dijelovima“ [45]. U ovome kontekstu često se dade vidjeti slogan “izvrši malo, izvrši često” (eng. *commit small, commit often*) pri čemu “izvrši” podrazumijeva čin predaje koda na sustav za verzioniranje, primjerice Git. Cilj je imati jedinstveni repozitorij koji sadrži kod čitave aplikacije, a koji je dijeljen među svim programerima koji rade na nj, kao što je bilo objašnjeno u prethodnom poglavlju Verzioniranje koda. Glavna ideja ovog pristupa jest ta da se promjene koje programeri rade nad kodom često izvršavaju. Primjerice, nakon što je programer promijenio boju gumba ili dodao neku manju funkcionalnost promjene se odmah trebaju spremiti i izvršiti. Samo izvršavanje koda na repozitorij pokrenuti će proces CI-ja koji će automatizirano testirati kod u raznim domenama, uskladiti kod sa pravilima pisanja koda za cijeli tim te pokrenuti druge procese potrebne za integraciju. Takav pristup pruža brojne prednosti, neke od kojih su:

- rano otkrivanje grešaka u kodu
- ubrzan razvojni ciklus aplikacije
- bolja suradnja među timovima

- smanjenje mogućnosti pogreške pri ručnoj integraciji dijelova aplikacije
- mogućnost izvođenja automatiziranih testova

Svaki od spomenutih argumenata zasebno već je dovoljan razlog za implementaciju CI/CD-ja u razvoju sljedeće web aplikacije, no proces ne bi bio potpun bez CD dijela, odnosno postavljanja aplikacije. CD ide korak dalje od CI-ja i automatizira postavljanje koda u produkcijsku okolinu. Drugim riječima, u ovome se koraku nastoje sve promjene u kodu primijeniti na produkcijsku aplikaciju. U vremenu prije CD-ja ovaj proces bi izgledao otprilike ovako, također, preskočiti ćemo neke dijelove kako su oni riješeni u CI-ju:

1. programer napravi promjene u kodu
2. programer spremi promjene lokalno na računalu
3. programer spremi promjene na sustav za verzioniranje
4. programer testira promjene
5. programer se ručno spoja na server gdje se poslužuje aplikacija
6. programer ručno prebacuje datoteke potrebne za nove promjene
7. programer izgradi novu aplikaciju
8. programer gasi staru aplikaciju i pokreće novu aplikaciju

Ovo je pojednostavljeni primjer koji ne ulazi predetaljno u postavljanje aplikacije, u stvarnome svijetu postoji još mnogo detalja na koje je potrebno pripaziti ili stvari koje je potrebno dodatno izvršiti. Usprkos tome, već u ovom pojednostavljenom primjeru dade se zamijetiti kako je proces spor, zamoran i ima mnogo mogućnosti za pogrešku. Nadalje, ovaj primjer polazi od pretpostavke da se aplikacija poslužuje samo na jednom serveru, no što ako ih ima dva, pet ili deset. Sada je ovaj proces potrebno ponoviti toliko puta i to za svaki puta kada se neku promjenu želi primijeniti. Ovoliko trošenja vremena često je dovodilo da se promjene izvršavaju u velikim skupovima koda, ili ne toliko često. Nasuprot tome, proces CD-ja izgledao bi otprilike ovako:

1. programer napravi promjene u kodu na svom računalu
2. programer spremi promjene lokalno
3. programer spremi promjene na sustavu za verzioniranje
4. novi kod automatizirano je testiran, poslan na server(e) i pokrenuta je nova verzija aplikacije

Ovim procesom smanjila se potreba programera da ručno izvršava dobar dio akcija za koje bi mu bilo potrebno poprilično vremena. Ovakav pristup razvoju aplikacija ima mnoge prednosti, kao što su:

- Pouzdano i brzo postavljanje aplikacije
- Konzistentni i automatizirani procesi
- Jednostavan povratak u prethodno stanje u slučaju grešaka
- Smanjenje vremena utrošenog od strane programera
- Brže dostavljanje novih funkcionalnosti korisnicima
- Nove verzije aplikacije je jednostavnije za popraviti
- Smanjen rizik grešaka aplikacije

Nakon što je definirano što CI/CD predstavlja i koje su mu prednosti, valja definirati i neke od alate koji ga omogućuju. U nastavku će biti objašnjeni oni alati za koje većina programera smatra da imaju najbolju integraciju sa postojećim sustavima te koji imaju minimalne probleme na koje se može naići tokom njihova korištenja.

6.6.2. CI/CD alati

U svrhu CI/CD-ja postoje brojni alati, neki od popularnijih su Jenkins, Gitlab CI ili Github Actions i jedan od novijih, Vercelov ugrađeni CI/CD proces. Svaki od ovih alata dolazi sa svojim prednostima i nedostatcima, no u osnovi rješavaju isti problem. Iz tog razloga odabir jednog od ovih alata najviše će ovisiti o trenutnoj arhitekturi koja je već odabrana.

Jenkins

Jenkins je jedan od vodećih automatizacijskih servera otvorenog koda. Predstavlja skalabilno rješenje koje zadovoljava potrebe i velikih kompanija. Svojim opsežnim sustavom priključaka predstavlja rješenje za gotove sve slučajeve i potrebe. Ipak, Jenkins nije idealno rješenje u svakom slučaju. U nastavku će biti izdvojeni glavne prednosti i nedostaci Jenkinsa generalno.

Tablica 12. Prednosti i nedostaci Jenkinsa (Izvor: Vlastita izrada prema [44])

Prednosti
- veliki sustav priključaka za gotovo sve potrebe - velika zajednica korisnika - omogućuje CI/CD za sve tehnologije - jednostavna konfiguracija u jednoj datoteci ili preko grafičkog sučelja
Nedostaci
- potreban je zaseban server i podosta resursa, što podiže cijenu održavanja - zahtjevna krivulja učenja

- dodatna tehnologija koju je potrebno održavati

Iako je Jenkins veoma moćan u smislu svojih mogućnosti, smatra se da zadaje dodatnu problematiku i često nije potreban. Prvo, potrebno je imati dodatan server kako, pogotovo na većim projektima, Jenkins zahtijeva podosta računalnih resursa. Drugo, iako je način konfiguracije relativno jednostavan, u jednoj je datoteci, zahtijeva visoku razinu znanja o Jenkinsu i serverima generalno kako bi mogao biti ispravan i siguran. Nadalje, iako su priključci jedna od boljih funkcionalnosti Jenkinsa radi fleksibilnosti koju omogućuju, često imaju sigurnosne propuste te moraju biti često ažurirani kako bi osigurali da sve radi kako je namijenjeno. Ipak, Jenkins ima svoje mjesto u razvoju aplikacija, ako već postoji prikladna infrastruktura za njegovo postavljanje ili je već implementiran, Jenkins predstavlja solidno rješenje.

Gitlab CI/CD

Gitlab CI/CD jedna je od Gitlabovih funkcionalnosti koja je integrirana sa sustavom za verzioniranje koda. Samim time omogućuje jednostavno povezivanje cijelog koda sa logikom za CI/CD koja je zapisana u datoteci naziva .gitlab-ci.yml. Gitlab automatski interpretira kod napisan u toj datoteci i kreira sve potrebne korake za izvršenje istih. Također, kako su programski kod i CI/CD logika usko povezani nije potrebno nikakvo povezivanje i/ili slanje podataka na vanjske servise ili servere, kao što je slučaj sa Jenkinskom.

Tablica 13. Prednosti i nedostaci Gitlab CI/CD-ja (Izvor: Vlastita izrada prema [45])

Prednosti
- jednostavna integracija sa programskim kodom (kolokacija) - ugrađen registar kontejnera za Docker - paralelno izvođenje izgradnji koda - skalabilnost sustava - automatizirani alati za provjeru sigurnosti koda
Nedostaci
- poteškoće radi kompleksnosti većih projekata - manja zajednica i manje priključaka naspram alternativnim tehnologijama - zahtijeva podosta resursa

Uzevši sve prednosti i nedostatke Gitlab CI/CD-ja u obzir smatra se da je dobar slučaj isključivo ako je Gitlab već implementiran i korišten za verzioniranje koda te ako pokriva slučajeve u domeni projekta, u suprotnome bolje je odabrati jednu od drugih spomenutih tehnologija.

Github Actions

Slično kao Gitlab CI/CD, Github Actions ugrađena je funkcionalnost platforme za verzioniranje koda Github. Omogućuje usku integraciju programskog koda sa logikom CI/CD-ja. Postoje brojni razlozi zašto koristiti Github Actions, a tek nekolicina protiv, kao što je vidljivo u tablici 14.

Tablica 14. Prednosti i nedostaci Github Actionsa (Izvor: Vlastita izrada prema [46])

Prednosti
- jednostavna integracija sa programskim kodom (kolokacija) - paralelno izvođenje izgradnji koda u raznim okruženjima - velika tržnica za besplatne priključke za gotovo sve potrebe - automatizirani alati za provjeru sigurnosti koda
Nedostaci
- poteškoće radi kompleksnosti većih projekata - ograničenja besplatnih mogućnosti servisa - na većoj skali potrebno je platiti - privatni repozitoriji nemaju pravo na besplatne akcije

Github Actions vjerojatno je najprikladnija od tehnologija za većinu primjena u sklopu CI/CD-ja. Jednostavnost implementacije, pregršt priključaka i uska integracija sa platformom za verzioniranje koda, koja se najvjerojatnije i koristi u tu svrhu, čine ga jednom od najidealnijih opcija. Jedan od nedostataka na koji pak valja obratiti pozornost jest činjenica da je Github Actions servis, stoga ovisno o zahtjevima i veličini projekta postoji mogućnost da se prekorači limit besplatnog izvršavanja akcija, nakon čega je potrebno plaćati za ovu uslugu. Ipak, smatra se da i unutar ograničenja postavljenih od strane Github Actionsa, ono rješava toliko problema i briga koje bi u suprotnom programeri morali dodatno rješavati.

Vercel

Posljednja od tehnologija za CI/CD je već spomenuta platforma od poduzeća Vercel. Iako je o Vercelu bilo pisano u sferi posluživanja aplikacija nije bio opisan proces od spremanja programskog koda do gotove aplikacije. Vercel omogućuje integraciju sa postojećim sustavima

za verzioniranje koda što uveliko pojednostavljuje proces i proširuje spektar tehnologija koje se mogu koristiti uz Vercel. Nadalje, preko intuitivnog sučelja odabiru se parametri i akcije koje je potrebno izvesti, a ostalo se odrađuje automatizirano. Kako je uočljivo iz ovog kratkog opisa, Vercel platforma ima veliku dobrobit za programere, ali jednako tako postoje određene stavke koje nisu idealne.

Tablica 15. Prednosti i nedostaci Vercela (Izvor: Vlastita izrada prema [47])

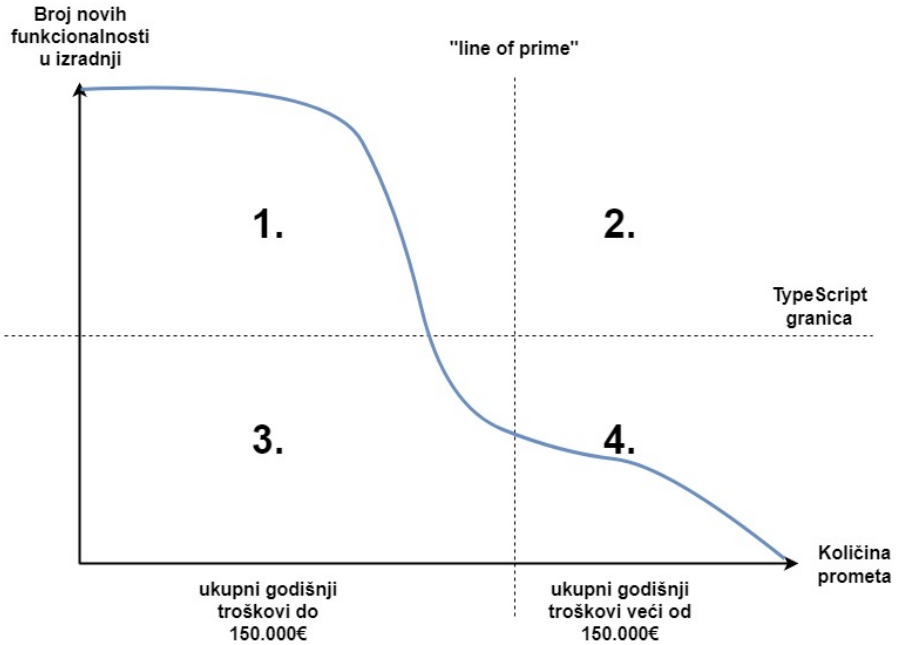
Prednosti
- jednostavna integracija sa programskim kodom u svega par klikova - automatizirana izgradnja i postavljanje aplikacije - testna, pretpregledna, i produkcijska okolina - suradnja unutra i između timova - CDN posluživanje diljem svijeta - spremanje svih izgradnji koda i jednostavno vraćanje verzija
Nedostaci
- visoke cijene - koristivo isključivo ako se Vercel koristi i za posluživanje aplikacije - uska povezanost sa Vercel platformom

Vercel platforma nudi najviše funkcionalnosti od svih dosad spomenutih tehnologija za CI/CD, štoviše, prema mišljenju velikog broja programera pruža najbolje korisničko iskustvo od svih sličnih tehnologija. Mogućnost integracije sa postojećim sustavom za verzioniranje koda pojednostavljuje cjelokupan proces CI/CD-ja. Nadalje, automatizirano posluživanje novih verzija aplikacije, kao i posluživanje testne i pretpregledne okoline gdje svaki programer sa pristupom može davati povratne informacije, samim čime se ubrzava i pojednostavljuje proces izrade novih funkcionalnosti te istovremeno osigurava njihova kvaliteta. Ipak, sve te pogodnosti imaju svoju cijenu, kako financijsku tako i vezanost za Vercel platformu. Za korištenje bilo koje od ovih pogodnosti potrebno je plaćati određenu svotu, dok neke mogućnosti nisu dostupne dokle se ne radi o većem poduzeću. Slično kao i kod Github Actionsa, smatra se kako kvaliteta usluga opravdava cijenu. Funkcionalnosti koje se dobivaju, efikasnost koju izazivaju kod programera te opće zadovoljstvo svih uključenih su neoborivi. Ipak, ovdje postoji određena granica cijene i koristi koju je potrebno uzeti u obzir, no više o tome biti će objašnjeno u narednom poglavlju.

7. Kada ne odabrati TypeScript

U životnom ciklusu svake aplikacije mogu se uočiti određeni trendovi. Kako aplikacije raste, tako se povećavaju zahtjevi, kako same aplikacije, tako i arhitekturalni zahtjevi. U cjelokupnom procesu bitno je znati kada i kako ispravno odabrati određene tehnologije. Iako je naziv ovog poglavlja “Kada ne izabrati TypeScript”, identičan princip vrijedi za sve druge tehnologije spominjane u prethodnim poglavljima. Iako su sve dosad spomenute tehnologije odlične za male i srednje projekte, postoji određena granica kada to više nije idealno. U programerskom žargonu ta se granica naziva „prvoklasna granica“ (eng. *line of prime*), iako bismo je u kontekstu ovog rada mogli nazvati i granicom za TypeScript.

Na narednoj ilustraciji, na slici 3, može se uočiti životni razvoj prosječne aplikacije, pri čemu se pretpostavlja da je u konstantnom porastu po broju korisnika. X os predstavlja ukupne godišnje troškove koji uključuju, ali nisu ograničeni na sve troškove održavanja aplikacije, osim plaće zaposlenika. Drugim riječima, to su troškovi posluživanja aplikacije, korištenih servisa i sl. Pretpostavimo da aplikacija u početku nema troškova te da oni rastu sa vremenom. Y os predstavlja broj funkcionalnosti koje je potrebno izraditi za aplikaciju. Plavo obojena krivulja predstavlja broj funkcionalnosti u izradi kako aplikacija postaje veća, odnosno ima veći promet. Nadalje, X i Y osi podijeljene su na dva dijela iscrtkanim crtama koje su nazvane “line of prime” i “TypeScript granica”. Te dvije iscrtkane crte istodobno dijele cijelu površinu grafa na četiri dijela, koji su označeni sa rednim brojevima od jedan do četiri radi lakšeg objašnjavanja.



Slika 3. “Line of prime” (Izvor: Vlastita izrada prema: [17])

Promatrajući graf sa slike 3 može se pratiti razvoj aplikacija ovisno o broj korisnika i funkcionalnosti koje je potrebno kreirati. U početcima razvoja aplikacije, pri vrhu X osi u 1. kvadrantu, potrebno je implementirati najviše funkcionalnosti. Ovo proizlazi iz osnovne činjenice kako aplikacija još ne postoji i potrebno je kreirati sve funkcionalnosti koje su zamišljene za prvotnu verziju aplikacije. Istodobno, pretpostavimo da koristimo plaćene servise poput spominjanog Vercela, u tom trenutku aplikacija još nema troškova ili ima minimalne troškove kako još nema korisnika. Kako aplikacija postaje popularnija, tako raste i broj korisnika i sukladno tome troškovi održavanja i posluživanja aplikacije. U procesu razvoja, aplikacija postaje stabilnija i bolje su definirane funkcionalnosti koje su uključene. Sve spomenute činjenice dovode do smanjenja potrebe implementacije novih funkcionalnosti te se fokus prebacuje na održavanje. Doduše, u cijelom spomenutom procesu postoji nekoliko granica na koje je potrebno obratiti pozornost, a koje možemo generalizirati u dvije kategorije, cijena i performanse.

Krenuvši od 1. kvadranta, može se uočiti kako su troškovi niski, a potrebno je izraditi mnogo funkcionalnosti, ovaj kvadrant idealan je za TypeScript aplikacije. Visoka brzina izmjena jedna je od TypeScriptovih glavnih prednosti, a još ne postoje visoke zahtjeve za performanse. Ovo je također prostor gdje se funkcionalnosti mogu izraditi i predstaviti korisnicima te ovisno o povratnim informacijama jednostavno mijenjati sve dok ne budu kvalitetne.

Suprotno spomenutim razlozima, ako se od početka zna da će se ubrzo doći u 3. kvadrant, zahtjevi se podosta mijenjaju. Treći kvadrant karakteriziraju aplikacije za koje su već od samog početka znane točno određene funkcionalnosti koje se neće previše mijenjati tokom života aplikacije. Primjer jedne takve aplikacije jest čitač za regularne izraze (eng. *regular expression*), skraćeno RegEx. Za RegEx postoje točno određeni zahtjevi i upute koje moraju biti zadovoljene u finalnom proizvodu. Iako se određene stavke mogu mijenjati, funkcionalnosti će uglavnom ostati iste, stoga je možda bolje razmotriti performantnije tehnologije od samog početka umjesto stvaranja gotove aplikacije i relativno brze potrebe prebacivanja u druge tehnologije.

Vrativši se na 1. kvadrant primjera rastuće aplikacije, kako troškovi aplikacije rastu aplikacija će doći do prvoklasne granice i prijeći u 4. kvadrant. Ovu granicu može se najjednostavnije definirati kao “granicu nakon koje je jeftinije uposliti više programera neko koristiti servise” [27]. Na početku ovog primjera pretpostavljeno je bilo da za posluživanje aplikacije koristimo Vercel platformu, koja je bila odabrana radi svih prednosti spomenutih u prethodnim poglavljima. Jedan od argumenata glasio je ovako: “Vercel je odabran kako bismo posao vezan uz infrastrukturu, CI/CD i sl. Taj posao prepustili Vercelu umjesto da se zaposle dodatni programeri”, što je validan argument. Ako se pretpostavi da je prosječna plaća jednog

backend programera 1.500€ [4] i da za implementaciju i održavanje barem osnovnih funkcionalnosti koje Vercel pruža potrebno barem 4 programera, sumirano na godišnjoj bazi to iznosi: $1.500€ * 12 * 4 = 72.000€$. Nadalje, iako se sve web aplikacije mogu izvesti sa TypeScriptom, u slučaju da aplikacija ima toliko visoke troškove da moramo uposliti nove programere, vrlo je vjerojatno da je aplikacija već stabilna. U tome slučaju isplativo je aplikaciju napisati u nekim od performantnijih programskih jezika kako bi se dodatno smanjili troškovi.

Iako je teško definirati neke jezike i/ili tehnologije koje bi mogle zadovoljiti potrebe svih aplikacija koje zahtijevaju nešto više od TypeScripta, postoji nekoliko popularnih opcija. Jedna od takvih jest programski jezik Golang, poznatiji prema njegovom skraćenom nazivu Go, koji je relativno sličan TypeScriptu. Ili pak nešto noviji jezik, Rust, čije ga performanse stavljaju na razinu C programskog jezika. Iako postoje još mnoge druge opcije, one izlaze van domene ovoga rada te kao takve neće biti objašnjene u dubinu.

8. Smjernice za izradu web aplikacija

U prethodnim poglavljima bili su objašnjeni svi koraci u razvoju aplikacije od početne ideje, odabira mogućih tehnologija i servisa ovisno o zahtjevima aplikacije, načinima kako pojednostaviti proces posluživanja aplikacija, sve do alata za praćenje stanja aplikacije i obrazloženja sa sve spomenute korake. U nastavku će svi dosad objašnjeni koraci biti sažeti u obliku tablice kako bi bilo jednostavnije snalaziti se u njima te kako bi mogli služiti kao predložak za odabir smjernica za izradu vlastite web aplikacije. Tablica je podijeljena u 4 stupca, prvi stupac služi samo za praćenje rednog broja smjernica, drugi stupac, „Korak“, definira sam naziv koraka. U trećem stupcu, „Ishod“, objašnjeno je što određena smjernica podrazumijeva i što je potrebno kreirati i/ili odabrati sa tom smjernicom. Posljednji stupac, „Mogućnosti/Pristup“ je opcionalan stupac u kojem su nabrojane tehnologije, servisi i/ili načini pristupa problemu koji mogu olakšati odabir korisniku ovih smjernica i određuju najbolji način kako realizirati „ishod“.

Tablica 15. Smjernice za izradu web aplikacija

	Korak	**Ishod**	**Mogućnosti/Pristup**
1.	Definirati ideju	Definirati (ne)funkcionalne zahtjeve aplikacije	
2.	Odabrati sustav za verzioniranje	Odabran sustav za verzioniranje koda	Gitlab, Github, BitBucket
3.	Definirati visoku razinu arhitekture aplikacije	Na temelju (ne)funkcionalnih zahtjeva imati definirane potrebne dijelove sustava	- ugrubo skicirati glavnu arhitekturu sustava - definirati sve servise koji će se koristiti i definirati granice između njih
4.	Odabrati bazu podataka	Odabrana baza podataka	MySQL, PostgreSQL, CouchDB, KeyDB, SQLite
5.	Odabrati platform u za posluživanje baze podataka	Odabran način posluživanja baze podataka	- samostalno posluživanje - baza podataka kao servis: PlanetScale, CouchDB,
4.	Odabrati Frontend tehnologiju	Odabrane(e) frontend tehnologija(e)	Vanilla HTML CSS JS, React, NextJS, SolidJS, Svelte, Astro
5.	Odabrati backend tehnologije	Odabrana(e) backend tehnologija(e)	NextJS, NodeJS, Fastify
6.	Odabrati platformu za posluživanje	Odabrana platforma za posluživanje aplikacije	- samoposluživanje - servisi: Vercel, Netlify, AWS, DigitalOcean

	aplikacije		
7.	Definirati potrebne servise (ako je primjenjivo)	Definirani ostali potrebni servisi	- jako ovisno o potreba aplikacije
8.	Definirati načine praćenja aplikacije	Odabrane tehnologije za praćenje i bilježenje sustava	- PostHog, Sentry, Elastic stack, Plausible
9.	Odabrati način implementacije CI/CD lanca	Implementiran CI/CD lanac	- samostalno kreiranje lanca: Github Actions, Jenkins, Gitlab CI/CD - servisi: Vercel, Netlify

Iako su smjernice u tablici 15. namijenjene da pokriju veći broj potencijalnih slučajeva aplikacije, valja napomenuti da određene odluke mogu imati problematične posljedice, ponajviše u pogledu potrebe za vlastitom implementacijom pojedinih servisa. Primjerice, ako se za posluživanje aplikacije odaberu druge platforme osim Vercela ili Netlifyja CI/CD lanac neće biti uključen. Samim time potrebno je pronaći način implementacije istog, što zahtijeva zakupljanje vlastitog servera i samostalne implementacije cijelog lanca. Iz toga razloga tablica 15. može biti pojednostavljena kako bi se osiguralo čim bolje programersko iskustvo i ubrzao razvoj aplikacije. Valja napomenuti kako su odabrane tehnologije i servisi odabrani prema empirijskoj analizi. Samim time odabiri su pristrani i možda neće moći pokriti sve slučajeve, no ako je ikako moguće preporučen je pristup kao što je prikazan u tablici 16.

Tablica 16. Skraćene smjernice za izradu web aplikacija

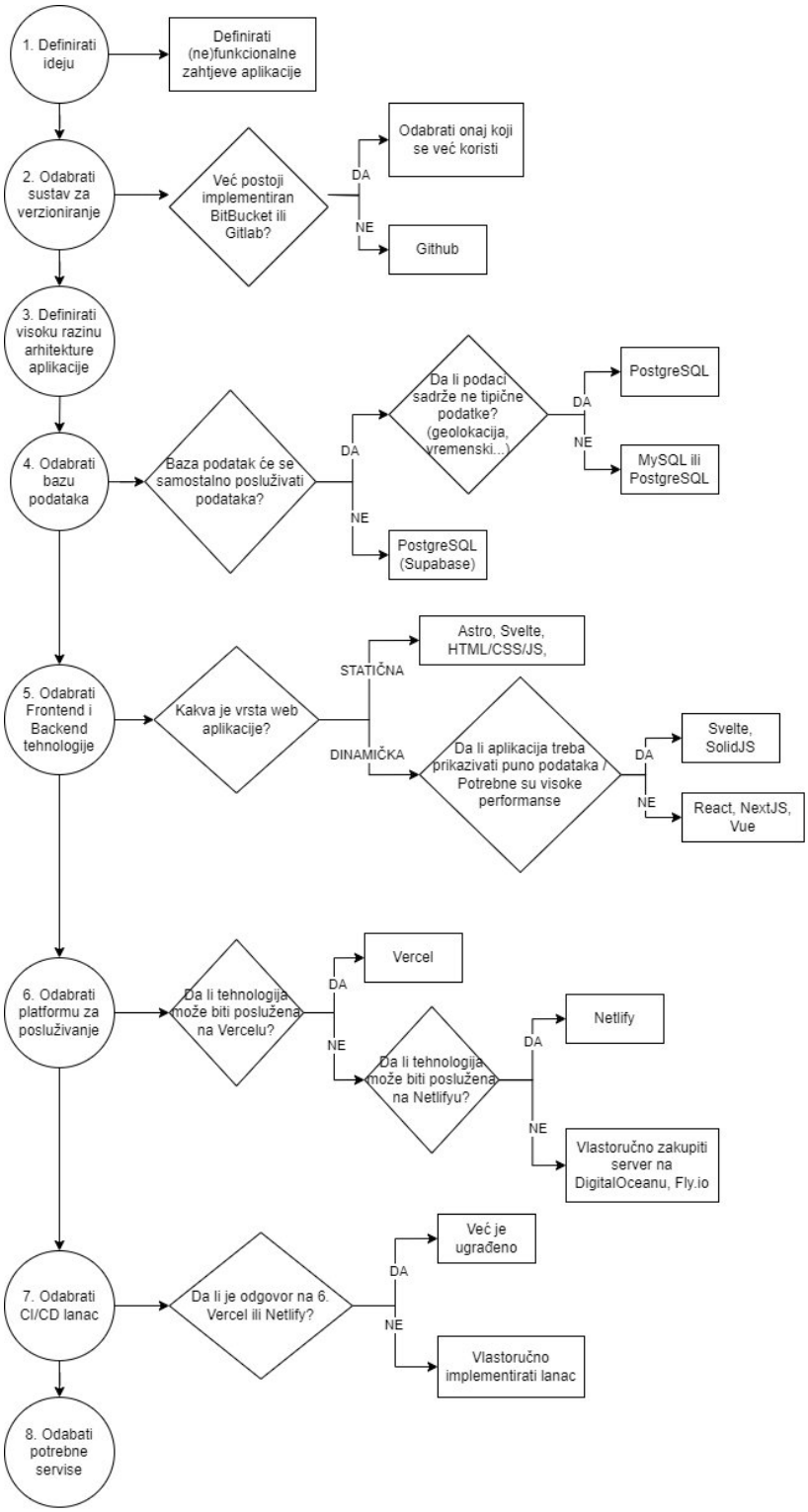
	Korak	**Ishod**	**Mogućnosti/Pristup**
1.	Definirati ideju	Definirati (ne)funkcionalne zahtjeve aplikacije	
2.	Odabrati sustav za verzioniranje	Odabran sustav za verzioniranje koda	Github
3.	Definirati visoku razinu arhitekture aplikacije	Na temelju (ne)funkcionalnih zahtjeva imati definirane potrebne dijelove sustava	- ugrubo skicirati glavnu arhitekturu sustava - definirati sve servise koji će se koristiti i definirati granice između njih
4.	Odabrati bazu podataka	Odabrana baza podataka	- PostgreSQL
5.	Odabrati platform u za posluživanje baze podataka	Odabran način posluživanja baze podataka	- baza podataka kao servis: Supabase

4.	Odabrati Frontend tehnologiju	Odabrane(e) frontend tehnologija(e)	Vanilla HTML CSS JS, React, NextJS, Solid, Svelte, Astro
5.	Odabrati backend tehnologije	Odabrana(e) backend tehnologija(e)	NextJS, NodeJS, Fastify
6.	Odabrati platformu za posluživanje aplikacije	Odabrana platforma za posluživanje aplikacije	- servisi: Vercel ako je odabrana frontend/backend tehnologija (NextJS, Svelte(kit), Nuxt, SolidStart)
7.	Definirati potrebne servise (ako je primjenjivo)	Definirani ostali potrebni servisi	- jako ovisno o potreba aplikacije
8.	Definirati načine praćenja aplikacije	Odabrane tehnologije za praćenje i bilježenje sustava	- PostHog, Sentry, Plausible
9.	Odabrati način implementacije CI/CD lanca	Implementiran CI/CD lanac	- Vercel, Netlify
10.	Razvoj aplikacije	Gotova aplikacije	
11.	Nadogradnja aplikacije	Nadograđena aplikacija	

Definiranje same ideje ne može se pojednostaviti samo tako jer je potrebno definirati sve potrebne (ne)funkcionalne zahtjeve. Stoga prva smjernica ostaje jednaka kao u proširenoj verziji. No razlike se dadu uočiti već u drugoj smjernici, odabiru sustava za verzioniranje. U većini slučajeva preporuča se Github radi svih spomenutih razloga u poglavlju o verzioniranju koda. Besplatan je, jednostavan za korištenje i ne zahtijeva pretjerano postavljanje od strane programera, jedino što je potrebno je kreirati račun te kreirati repozitorij za projekt. Nadalje, izrada visoke razine arhitekture sustava također ostaje jednaka. Bez obzira na projekt ovaj korak je neizostavan jer svim sudionicima na projektu daje uvid kako će sustav biti izrađen. Za bazu podataka, ako je potrebna, preporučeno je korištenje PostgreSQL-a kako obuhvaća najveći spektar mogućnosti. Uz klasične tipove podataka i relacijske podatke, moguće je kreirati temporalne, nestrukturirane podatke i dr. Jedini potencijalni razlog za ne korištenje PostgreSQL tehnologije jesu bolje performanse koje se mogu dobiti korištenjem drugačije vrste baze podataka. Ipak, takve optimizacije moguće je uraditi i naknadno, kada sustav to zahtijeva. Kao platformu za posluživanje baze podataka preporuča se Supabase, servis koje je pouzdan ima širok spektar funkcionalnosti, pristupačan platni model te podržava PostgreSQL koji je bio izabran u prethodnom koraku. Za razliku od prethodnih smjernica gdje je uglavnom bila odabrana isključivo jedna mogućnost, odabir frontend i backend tehnologija

nešto je kompleksniji. Prvo je potrebno postaviti pitanje, da li je potreban samo frontend ili i backend. Drugim riječima, hoće li web aplikacija biti uglavnom statična ili će se podaci morati moći spremati i uređivati. U slučaju da aplikacija treba biti dinamička, sljedeće pitanje jest, da li se može, ili želi, odabrati fullstack tehnologija ili će frontend i backend biti odvojeni. Ovo pitanje uglavnom ovisi o željama programera jer se većina problema može riješiti na oba načina. Primjerice, ako se odluči za korištenje fullstack tehnologije, NextJS bio bi odgovor za smjernicu 4 i 6. Ako se pak odluči za odvojene tehnologije, jedna od mogućnosti bila bi čisti HTML, CSS i JS za frontend te Fastify za backend. Valja ponoviti da je odabir ovih tehnologija proizvoljan, iako će u određenim prilikama jedna od mogućnosti biti nešto bolja od drugih, u svima će se moći kreirati funkcionalna aplikacija. Ako ponovo promotrimo tablicu 2. i poglavlje 5.3. mogu se uočiti neke od prilika gdje jedna tehnologija prevladava nad drugima. Kao posljedica odabira frontend i backend tehnolgija nadovezuje se odabir poslužitelja web aplikacije. Može biti nazvano posljedicom jer su određeni poslužitelji namijenjeni samo za određene tehnologije. Ipak, sa mogućnostima danim u smjernici 6 najpogodnija i najjednostavnija opcija jest Vercel. Smjernicu 7 također je teško pojednostaviti, kako odabir servisa najviše ovisi o zahtjevima aplikacije. Primjerice, ako je potrebno ugraditi neke aspekte umjetne inteligencije može se odabrati aktualni ChatGPT, ako je pak potrebno imati obostranu komunikaciju svih povezanih klijenata u realnom vremenu, mogao bi se koristiti servis Pusher, itd. Samim time ova smjernica prepušta se na odabir programera, odnosno arhitekta sustava. Smjernica 8 dotiče se praćenja i bilježenja sustava, iako je bilo navedeno mnogo opcija empirijski je zaključeno da je optimalna kombinacija Sentry za praćenje grešaka u kodu i dobivanje povratnih informacija, Plausible za praćenje generalnih metrika aplikacije, primjerice broja korisnika te PostHog za detaljno bilježenje događaja unutar aplikacije. Pri posljednjoj smjernici 8, potrebno je odabrati CI/CD lanac, odnosno način implementacije istog. Iako je ova smjernica posljednja, uvelike je definirana prethodnim odabirima. Kako je u smjernici 2 odabran Github kao sustav za verzioniranje potreban je CI/CD lanac koji funkcionira sa njime, a kako je u smjernici 6 odabran Vercel za posluživanje aplikacije koji ujedno ima ugrađen CI/CD lanac, ne postoji razlog zašto ga ne odabrati. Samim time pojednostavljena je i nadogradnja aplikacije. Kako je Vercel integriran u CI/CD proces, može pratiti svaku promjenu koja se desi na Githubu. U klasičnom toku razvoja, programeri promjene stavljaju na novu granu u sustavu za verzioniranje, koja se nakon provjere spaja na glavnu granu i postaje najažurnija verzija aplikacije [70]. Vercelova infrastruktura detektirane promjene na glavnoj grani automatski pokreće i u svega par minuta najnovija verzija aplikacija isporučena je. Dodatno, svaka razvojna grana poslužuje svoju pretpreglednu verziju aplikacije na kojoj je moguće testirati i pregledati sve promjene.

Ovim pojednostavljivanjem smanjila se mogućnosti odabira, no samim time je ubrzan proces odabira. Ipak, kako je teško prikazati sve moguće varijable i potencijalne rizike prilikom odabira svakog od navedenih koraka, isti će se prikazati u obliku slijednog grafa na slici 4 koji će jednostavnije ilustrirati odluke koje je potrebno dovesti.



Slika 4. Prikaz smjernica za razvoj aplikacije u obliku grafa odlučivanja (Izvor: Vlastita izrada)

Kao što se dade uočiti iz grafa sa slike 4, nadodane su dodatne varijable o kojima je potrebno voditi računa pri odabiru tehnologija. Jedna od tih javlja se već kod odabira sustava za verzioniranje. Iako je u skraćenim smjernicama u tablici 16 bilo predloženo da se koristi isključivo Github, moguće je odabrati alternative. Ipak, ovaj izbor uglavnom ovisi o tome da li već postoji implementirano drugo rješenje. Nadalje, dan je bolji prikaz za odabir tehnologija baze podataka kao i nešto detaljniji pregled odabira frontend i backend tehnologija. Također, smjernica 7 prikladno ukazuje na ovisnost o prethodnim odabirima.

Iako je gotovo nemoguće stvoriti jedinstven set rješenja za kreiranje bilo kakve web aplikacije, predstavljenih 9 smjernica pokušava postići ravnotežu između dovoljne apstrakcije koja pokriva čim veći spektar slučajeva izrade web aplikacija sa TypeScriptom, konkretnog odabira potrebnih tehnologija te koraka u razvoju. Kao takve, definirane smjernice predstavljaju dobru početnu točku koja može olakšati bitne odluke pri razvoju web aplikacija. U ostatku ovoga rada biti će prikazano korištenje definiranih smjernica na realnom primjeru kako bi mogla biti prikazana njihova korist u procesu razvoja web aplikacija.

9. Izrada web aplikacije prema smjernicama

Implementacija same aplikacije pratiti će prethodno definirane smjernice kako bi mogao biti prikazan praktični pristup razvoju aplikacija. Aplikacija će predstavljati platformu za vođenje i praćenje projekata, što je u skladu sa temom rada.

9.1. Primjena smjernica za razvoj aplikacije

Kako je bilo definirano prvom smjernicom u tablici 16, prvi korak razvoja web aplikacije jest definirati samu ideju. Aplikacija „Project planner“ nastoji pojednostaviti način na koji individualni programeri i timovi prate svoje projekte. Pomoću intuitivnog sučelja i naprednih mogućnost, ova aplikacija omogućuje korisnicima da efikasno organiziraju i prate vlastite projekta od početne ideje do krajnjeg proizvoda.

Jednom definiranu ideja potrebno je raspisati kao (ne)funkcionalne zahtjeve, kako bi se pojednostavilo praćenje toka razvoja aplikacije i kako bi se ideja konkretizirala u korake stvarne implementacije.

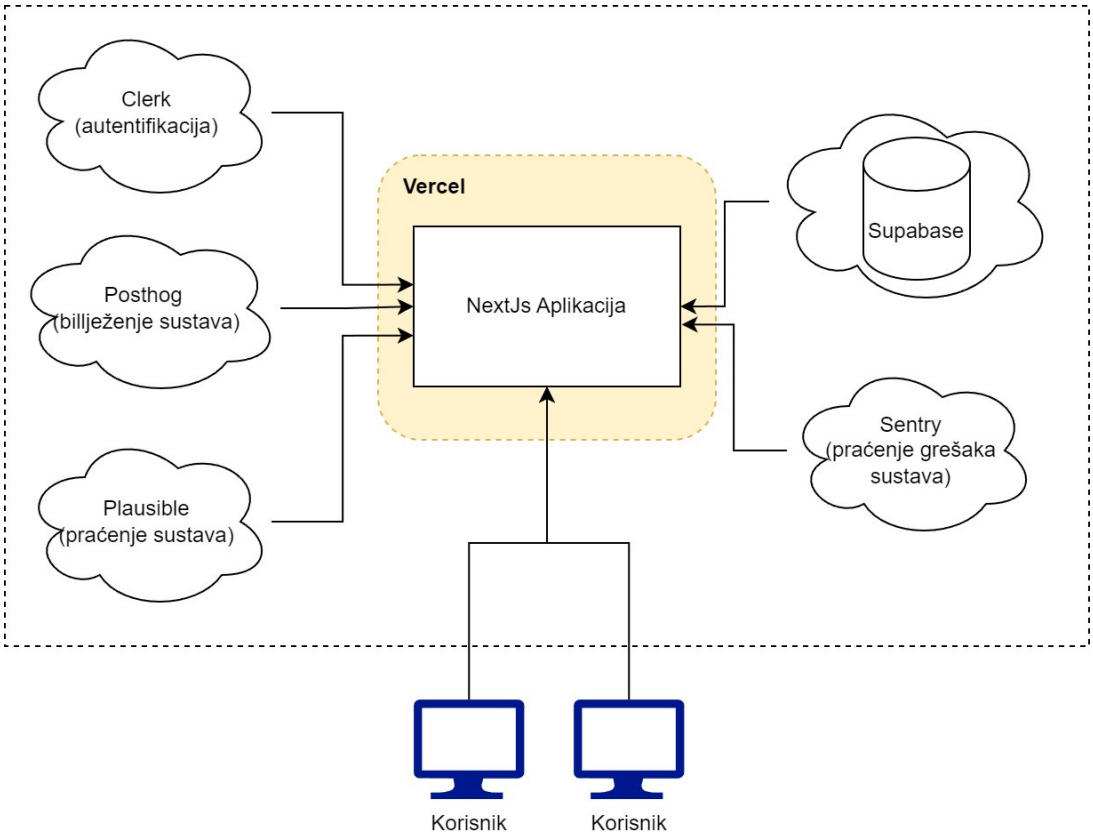
Tablica 17. Primjer definiranja funkcionalnih zahtjeva

Oznaka	**Naziv**	**Opis**	**Bitno**
F01	Prijava	Kao korisnik želim se moći prijaviti u aplikaciju.	1
F02	Kreiranje projekta	Kao korisnik želim moći kreirati novi projekt.	1
F03	Pregled projekata	Kao korisnik želim moći imati pregled svih svojih projekata i projekata u kojima sam član.	1
F04	Upravljanje projektom	Kao vlasnik projekta želim moći uređivati kreirani projekt	2
F05	Upravljanje članovima projekta	Kao vlasnik projekta želim moći dodavati i ukljanjati druge korisnike u projekt.	3
F06	Upravljanje zadacima	Kao član projekta želim moći dodavati nove zadatke za projekt	3
NF01	Pouzdanost aplikacije	Aplikacija treba moći podržati istovremeno korištenje do 10.000 korisnika.	
NF02	Sigurnost	Aplikacija mora biti sigurna. Aplikaciju mogu koristiti samo prijavljeni korisnici.	

Za sljedeći korak, uzevši u obzir argumente definirane u poglavlju 6.3.6, najbolji sustav za verzioniranje je Github. Razlog tomu jest činjenica da ne postoji unaprijed određen sustav za verzioniranje, a Github je besplatan i nudi najširi spektar funkcionalnosti. U narednom koraku, kako bi se definirala visoka razina arhitekture aplikacije, potrebno je donijeti odluke od kojih dijelova se treba sastojati. Kako aplikacija zahtijeva da podaci budu spremani, moraju se moći kreirati projekti, korisnici i zadaci, potrebno je odabrati neku bazu podataka. Radi jednostavnosti, za svrhu ovog projekta najbolje je odabrati neku bazu podataka koju je moguće koristiti kao servis, kako tada nije potrebno imati vlastiti server za posluživanje baze podataka. Nadalje, kako će aplikacija trebati moći pratiti korisnike, svaki od kojih može kreirati svoje projekte, dodavati druge korisnike u projekt te kreirati zadatke za taj projekt, uočljivo je da su podaci visoko povezani. Prema tome, relacijski model tehnologija baze podataka idealan je za takav scenarij. Jedan od servisa koji odgovara ovim kriterijima jest Supabase koji se temelji na PostgreSQLu [66]. Uključeno je darežljivo besplatno ograničenje, visoka skalabilnost, pouzdanost i sigurnost. Sljedeći koraci uključuju odabir frontend i backend tehnologija te platforme za posluživanje. Kako su sve one povezane, ovisno o nekim platformama za posluživanje nije moguće koristiti sve frontend ili backend tehnologije, sagledane su zajedno. Prema (ne)funkcionalnim zahtjevima dade se zaključiti kako aplikacija ne iziskuje određeni jezik jer je problem relativno trivijalan i svodi se na klasičnu aplikaciju sa sučeljem za upravljanjem podataka, tj. CRUD aplikaciju (eng. *Create Read Update Delete*). Iz tog razloga, posluživanje aplikacije generalniji je problem i može određivati koje tehnologije je moguće koristiti te je bolje njega prvo riješiti. Prema saznanjima iz poglavlja 6.4. može se odrediti koji je način posluživanja aplikacije najidealniji. Iz jednakih razloga kao pri odabiru načina posluživanja aplikacije, jednostavnosti i pouzdanosti, najbitniji su aspekti na koje valja obratiti pozornost. Samim time, najadekvatniji pristup jest platforma Vercel, koja također rješava problematiku CI/CD lanca i integrirana je sa Githubom, odabranim kao sustavom za verzioniranje. Iako je dio problema riješen odabirom Vercela, frontend i backend tehnologije koje sada mogu odabrati ograničenije su. Prema ideji aplikacije može se zaključiti kako će aplikacija biti interaktivna, korisnici će moći dodavati nove projekte, dodavati zadatke, upravljati zadacima i sl. Iako ova konkretna činjenica nije dovoljno ekstremna kako bi zahtijevala specifičnu tehnologiju, valja uzeti u obzir da ju je potrebno razmotriti. Iz svega zaključenog slijedi da odabir frontend i backend ili pak fullstack tehnologije ne igra preveliku ulogu. U tome slučaju odabir tih tehnologija preostaje na programeru(ima), uz jedno ograničenje, da se može posluživati na Vercelu. Kako je odlučeno da je Vercel platforma korištena za posluživanje i odvajanje backend i frontend tehnologija nije nužno, prema poglavljima 6.3.1, 6.3.2 i 6.3.3, odabran je NextJs. NextJs kao razvojni okvir u jednom projektu sadržava frontend i backend, dodatno razvijen je od strane Vercela i ima prvoklasnu podršku za platformu i kao takav je odličan odabir.

Za naredni korak, odabir servisa, ponovo valja se uzeti u obzir koji su funkcionalni zahtjevi aplikacije. Većina zahtjeva može riješiti samom bazom podataka. Projekte koji se trebaju kreirati, korisnici koji se trebaju spremati, kao i zadaci svi se spremaju i čitaju iz baze podataka. Jedina funkcionalnost koja iziskuje poveću pažnju jest prijava, odnosno autentifikacija. Kvalitetno i sigurno odraditi autentifikaciju nije tako jednostavno, stoga su istražene trenutno korištene biblioteke i servisi u TypeScript ekosustavu koji se koriste u tu svrhu. Pronađeno je par rješenja: NextAuth.js, Lucia auth i Clerk. Prvospomenuti NextAuth.js je biblioteka otvorenog koda za autentifikaciju za NextJs razvojni okvir [67]. Od svih spomenutih ono je najzrelija biblioteka, za razliku od naredna biblioteke, Lucia auth, koja je najnovija je od tri spomenute. Lucia auth također je biblioteka otvorenog koda, no činjenica koje je izdvaja u odnosu na NextAuth.js jest ta da je neovisna o razvojnom okviru (eng. *Framerwork-agnostic*), odnosno, može se koristiti sa čistim JavaScriptom, TypeScriptom ili bilo kojom drugo tehnologijom temeljenom ja JavaScriptu [68]. Clerk je specifičan prema tome što je servis za autentifikaciju. Iako je relativno novi servis, brzo je postao korišten od velikog broja individualnih programera, kao i poduzeća, jedan od kojih je primjerice Stripe. Dodatno, ima ugrađenu integraciju sa prethodno odabranim servisom za posluživanje baze podataka, SupaBaseom [69]. Polazivši od načela jednostavnosti kao do sada, idealan odabir za autentifikaciju predstavlja servis Clerk koji garantira sigurnost same aplikacije. Posljednji odabir potencijalnih servisa moguće je definirati u pogledu praćenja i bilježenja aplikacije. Kako je bilo objašnjeno u poglavlju 6.5 i definirano u smjernicama iz tablice 16, najbolji odabir jesu Posthog i Plausible, specifično njihova verzija kao servis.

Sa svim definiraim tehnologijama i servisima sada je moguće dizajnirati visoku arhitekturu aplikacije kao što je vidljiva na slici 5.



Slika 5. Visoka arhitektura aplikacije Project planner (Izvor: Vlastita izrada)

Sa definiranom visokom arhitekturom aplikacije odrađeni su svi koraci određeni smjernicama iz tablice 16. Radi preglednosti valja sve odabirae zapisati u novu tablicu kreiranu sa tablicom 16 kao predloškom. Rezultat je tablica 18, kao što je vidljiva u nastavku.

Tablica 18. Primjena smjernica za izradu web aplikacija za aplikaciju „Project planner“

	Korak	**Rezultat**
1.	Definirati ideju	Aplikacija „Project planner“ nastoji pojednostaviti način na koji individualni programeri i timovi prate svoje projekte. Pomoću intuitivnog sučelja i naprednih mogućnost, ova aplikacija omogućuje korisnicima da efikasno organiziraju i prate vlastite projekta od početne ideje do krajnjeg proizvoda.
2.	Odabrati sustav za verzioniranje	Github
3.	Definirati visoku razinu arhitekture aplikacije	Promotriti sliku 5
4.	Odabrati bazu podataka	PostgreSQL

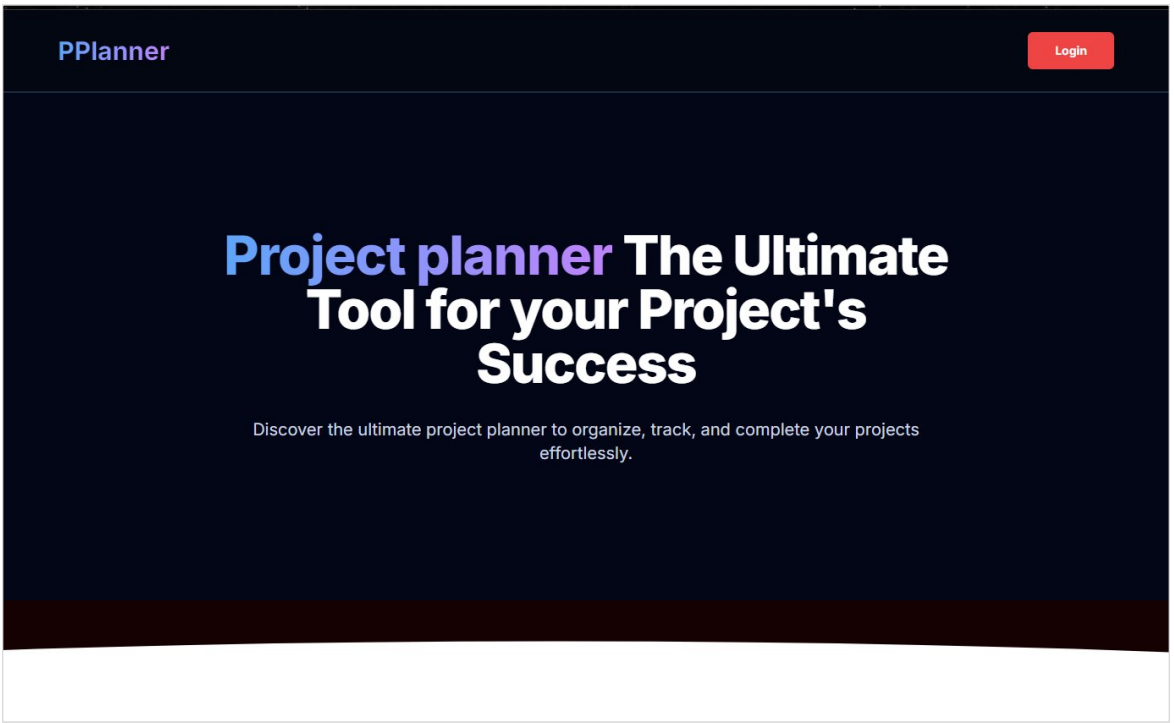
5.	Odabrati platform u za posluživanje baze podataka	Supabase
4.	Odabrati Frontend tehnologiju	NextJS
5.	Odabrati backend tehnologije	NextJS
6.	Odabrati platformu za posluživanje aplikacije	Vercel
7.	Definirati potrebne servise (ako je primjenjivo)	Clerk, (Posthog, Plausible, Sentry)
8.	Definirati načine praćenja aplikacije	Posthog, Sentry
9.	Odabrati način implementacije CI/CD lanca	Vercel
10.	Razvoj aplikacije	
11.	Nadogradnja aplikacije	

Završno sa tablicom 18, odrađeni su i definirani svi koraci u smjernicama za razvoj web aplikacija sa TypeScriptom. Odabrani su svi kritični dijelovi aplikacije i jedino što preostaje jest implementacija same aplikacije i potencijalne nadogradnje

9.2. Prikaz rada aplikacije

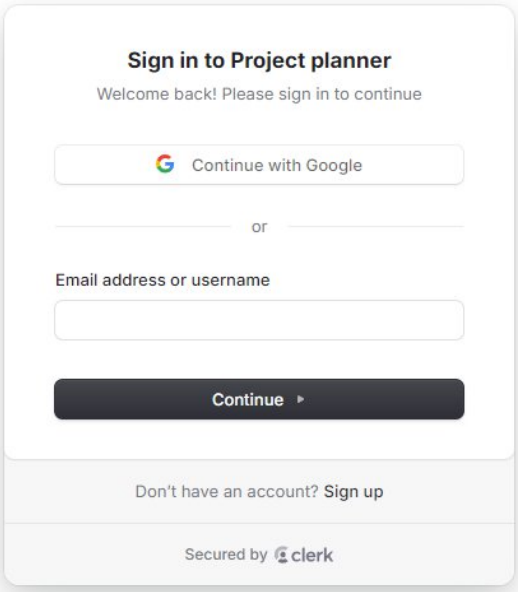
Aplikacija project planner nastoji pojednostaviti vođenje projekata korisnika, radi čega je sučelje dizajnirano na principu jednostavnosti i pristupačnosti. Kako je bilo definirano u početnim dijelovima ovoga rada, cillj izrade aplikacije u ovoj fazi jest kreiranje MVPa, drugim riječima potrebno je kreirati sve funkcionalnosti iako potencijalno nije u potpunosti definirano sučelje i nisu razrađeni svi rubni slučajevi.

Kada korisnik otvori početnu stranicu dočekuje ga kratak opis aplikacije Project planner i mogućnost za prijavu, što je vidljivo na slici 6.



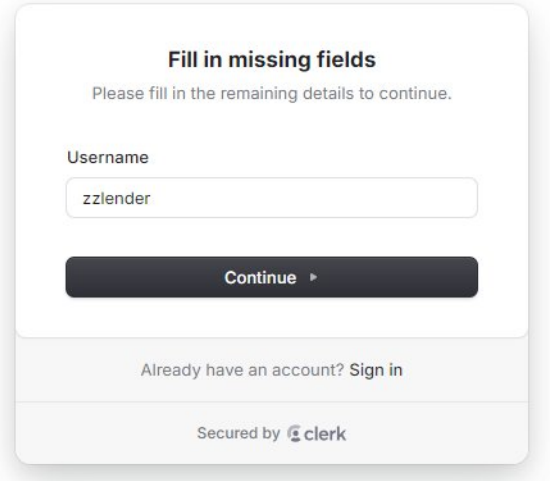
Slika 6. Početna stranica aplikacije Project planner (Izvor: Vlastita izrada)

Klikom na gumb za prijavu (eng. *Login*) korisniku se otvara stranica za prijavu i registraciju pri čemu se može registrirati i/ili prijaviti pomoću email adrese ili Google računa. Na slici 7 prikazana je spomenuta stranica. Prikazana komponenta za prijavu i registraciju sastavni je dio Clerk servisa definira se jednostavnim kopiranjem iz dokumentacije o korištenju Clerka.



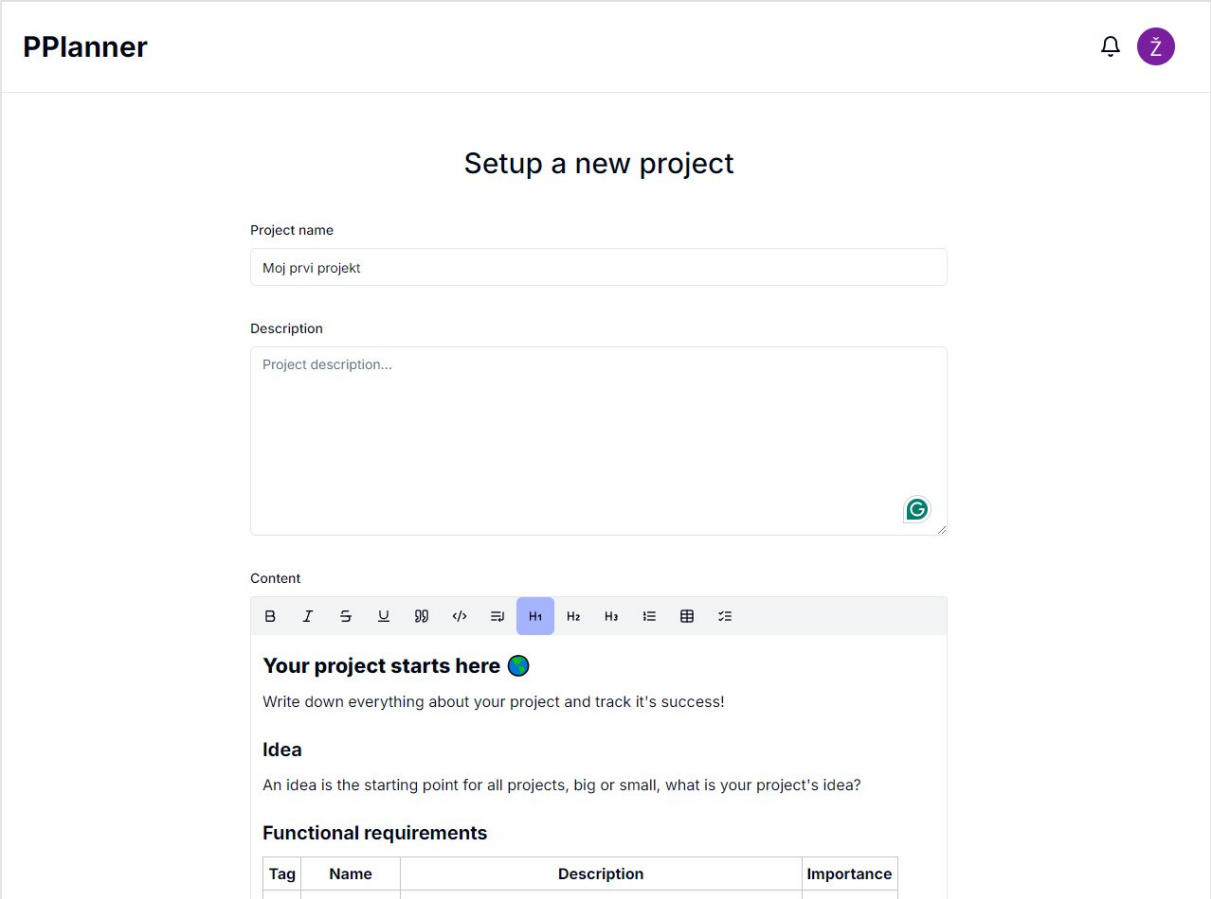
Slika 7. Stranica za prijavu aplikacije Project planner (Izvor: Vlastita izrada)

Pri registraciji, novi korisnici obavezni su unijeti jedinstveno korisničko ime koje će biti prikazivano drugim korisnicima i kako će moći, primjerice, pozivati druge korisnike u projekte. Na narednoj slici 8 prikaz je izgled sučelja za unos korisničko imena, koji je kao i sučelje za prijavu, dan od strane Clerka.



Slika 8. Prikaz unosa korisničko imena aplikacije Project planner (Izvor: Vlastita izrada)

Jednom prijavljen, svaki korisnik može kreirati vlastite projekte klikom na gumb Kreiraj novi projekt (eng. *Create new project*). Pri tome otvara se stranica za unos projekta kao što je prikazana na slici 9. Potrebno je unijeti naziv i opis projekta, dok su sve ostale informacije nepotrebno kod samog kreiranja. Za unos svih drugih informacija korišten je uređivač teksta koji podržava uređivanje teksta, stilizirane naslove, umetanje tablica i dr.

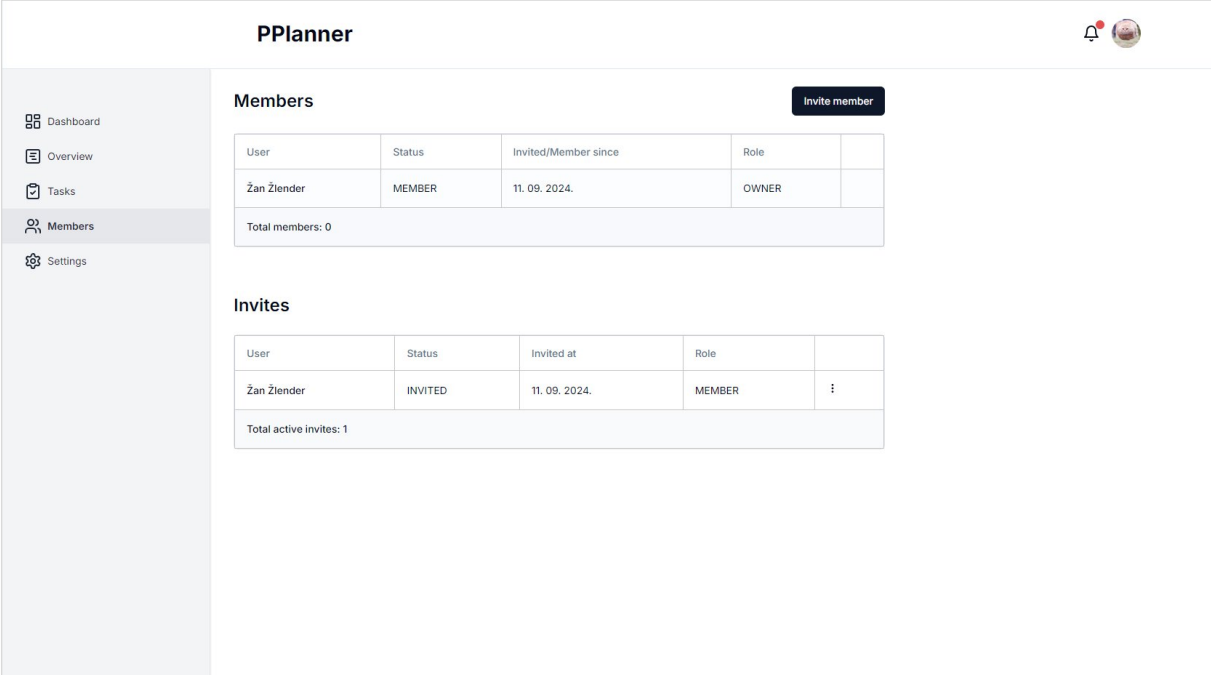


Slika 9. Stranica za kreiranje projekta aplikacije Project planner (Izvor: Vlastita izrada)

Svi projekti čiji je autor prijavljeni korisnik ili oni u kojima je trenutni korisnik član, prikazani su na početnoj upravljačkoj ploči (eng. *Dashboard*), kao što je vidljivo na slici 10.

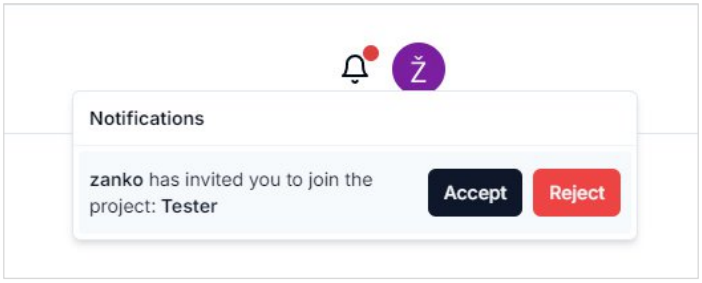
Slika 10. Prikaz svih projekata korisnika aplikacije Project planner (Izvor: Vlastita izrada)

Projekt je moguće pregledati ako korisnik ima ispravna prava, odnosno ako je autor samog projekta ili član, dok isključivo autor može uređivati informacije o projektu. Klikom na gumb Članovi (eng. *Members*) vlasnik projekta može dodavati ljude u projekt i upravljati pozivima u projekt. Sučelje za upravljanje članovima prikazano je na slici 11.



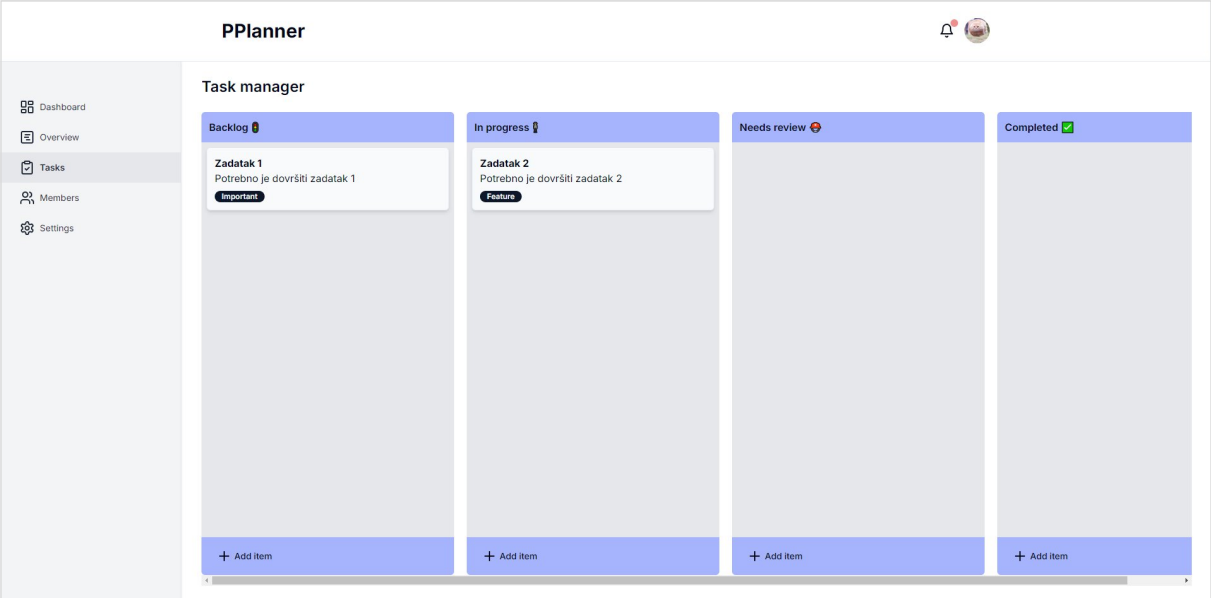
Slika 11. Stranica za upravljanje članovima aplikacije Project planner (Izvor: Vlastita izrada)

S druge strane, pozvani korisnik dobiva notifikaciju kada mu je poslana pozivnica u projekt te je može potvrditi ili odbiti, što je ilustrirano slikom 12.



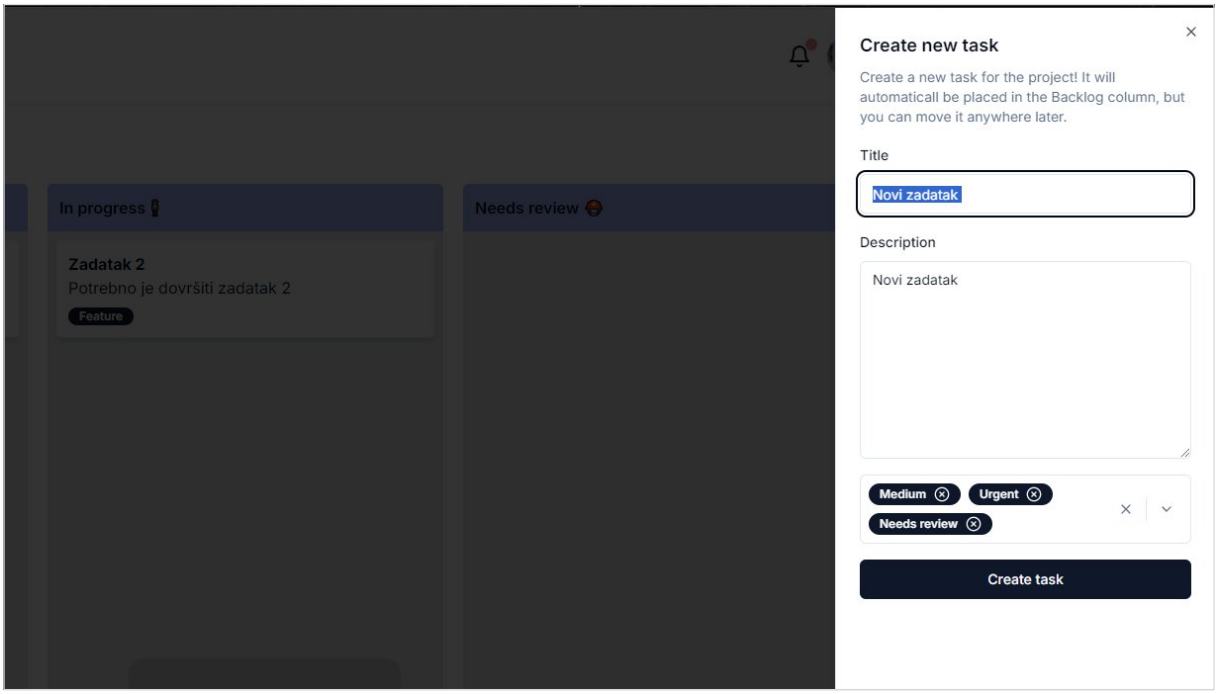
Slika 12. Prikaz poziva u projekt aplikacije Project planner (Izvor: Vlastita izrada)

Nadalje, jedna od najnaprednijih mogućnosti ove aplikacije jest upravljanje zadacima. Unutar samog projekta, klikom na gumb Zadaci (eng. *Tasks*) prikazuje se Kanban ploča u koju je moguće dodavati zadatke u jedan od definiranih stupaca. Slika 13 prikaz je spomenute Kanban ploče sa dva unesena zadatka, jedan u stupac „Popis zadataka“ (eng. *Backlog*) i jedan u stupac „U tijeku“ (eng. *In progress*). Sami stupci definirani su prema klasičnom Github Flowu, pri čemu postoje stupci za popis zadataka, zadatke u tijeku, zadatke koje je potrebno provjeriti i završene zadatke.



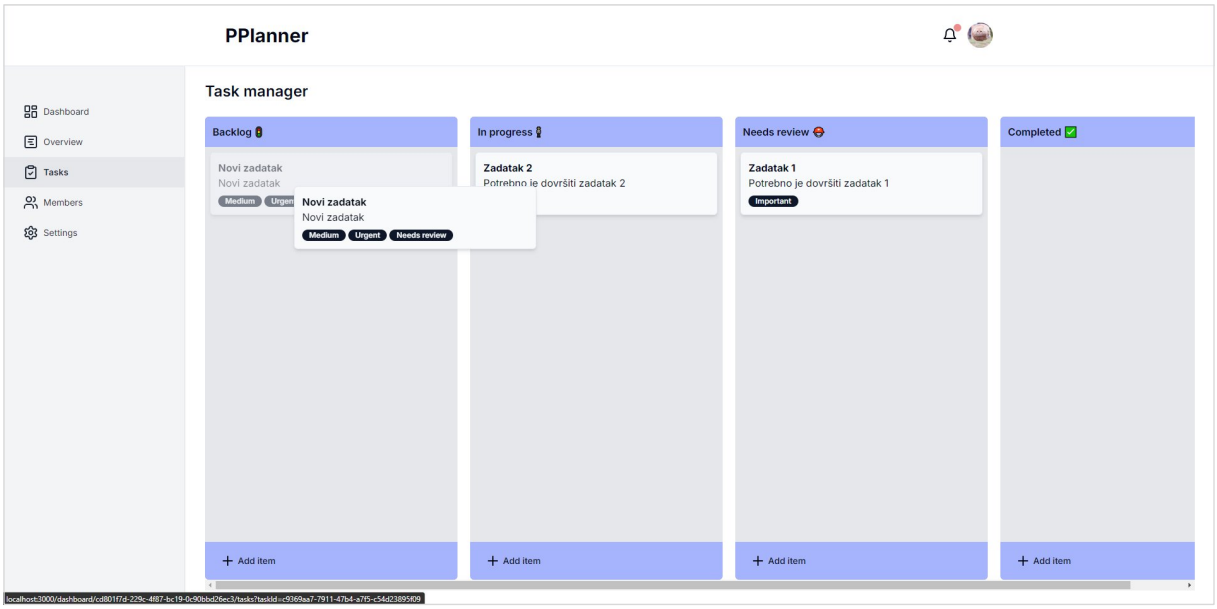
Slika 13. Stranica zadataka aplikacije Project planner (Izvor: Vlastita izrada)

Klikom na gumb Dodaj stavku (eng. *Add item*)“ otvara se klizni prozor za dodavanje novog zadatka, kao na slici 14, pri čemu je potrebno unijeti naziv zadatka, kratak opis i opcionalno dodati medalje (eng. *Badges*).



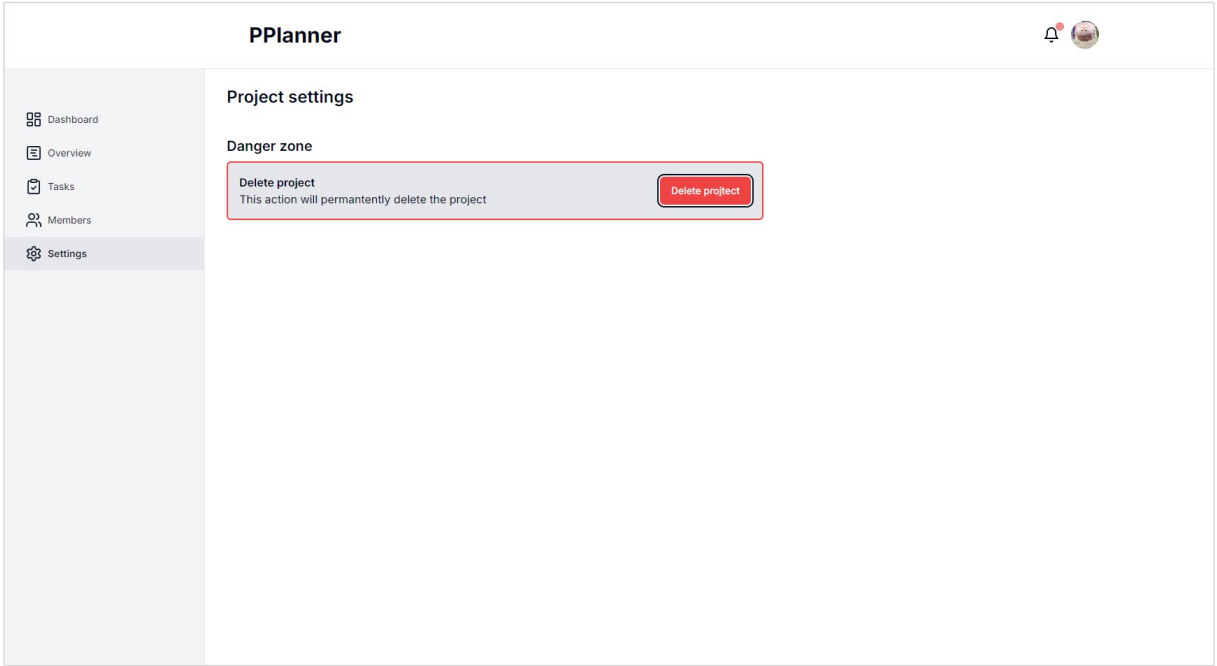
Slika 14. Dodavanje novog zadatka aplikacije Project planner (Izvor: Vlastita izrada)

Zadaci se mogu uređivati, proizvoljno pomicati iz jedne kolone u drugu klikom i povlačenjem pojedinog zadatka te se na jednak način mogu sortirati unutar kolone. Na slici 15 prikazana je mogućnost pomicanja, tokom pomicanja zadatka.



Slika 15. Sortiranje zadataka aplikacije Project planner (Izvor: Vlastita izrada)

Finalno, klikom na gumb Postavke (eng. *Settings*) dostupne su akcije vezane uz projekt, što je prikazano slikom 16. Tako je projekt moguće obrisati pri čemu se brišu svi podaci o projektu.

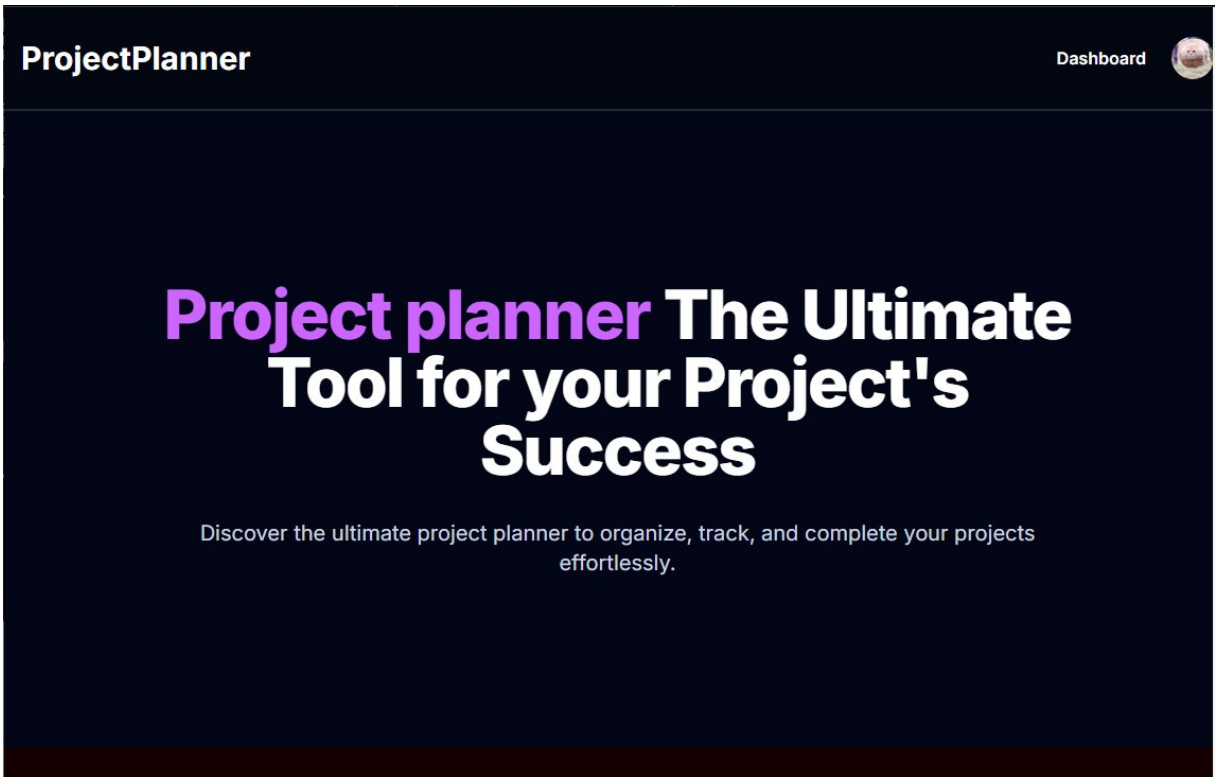


Slika 16. Postavke projekta aplikacije Project planner (Izvor: Vlastita izrada)

9.3. Nadogradnja aplikacije

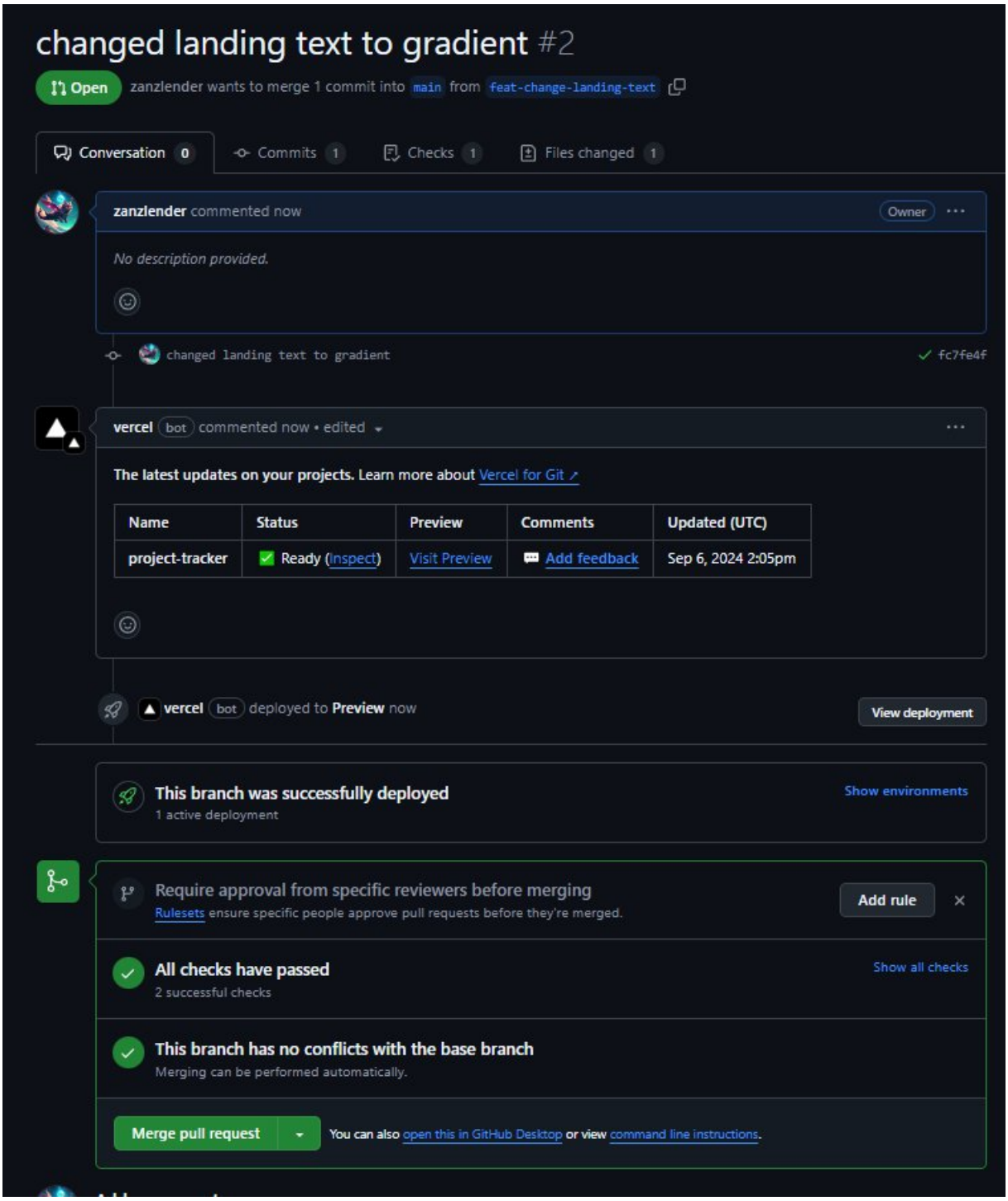
Zadnji korak ciklusa razvoja aplikacije kojeg valja spomenuti jest nadogradnja aplikacije. Kako je bilo ukratko objašnjeno u poglavlju 8, Vercel je integriraran sa CI/CD lancem

i detektira promjene na granama unutar repozitorija. Najjednostavije jest objasniti ovaj korak praktičnim primjerom. Pretpostavimo jedan primjer nadogradnje aplikacije, želi se promjeniti boja u naslovu početne stranice prikazane na narednoj slici. Umjesto ljubičaste boje potrebno je staviti plavo-ljubičasti gradijent.



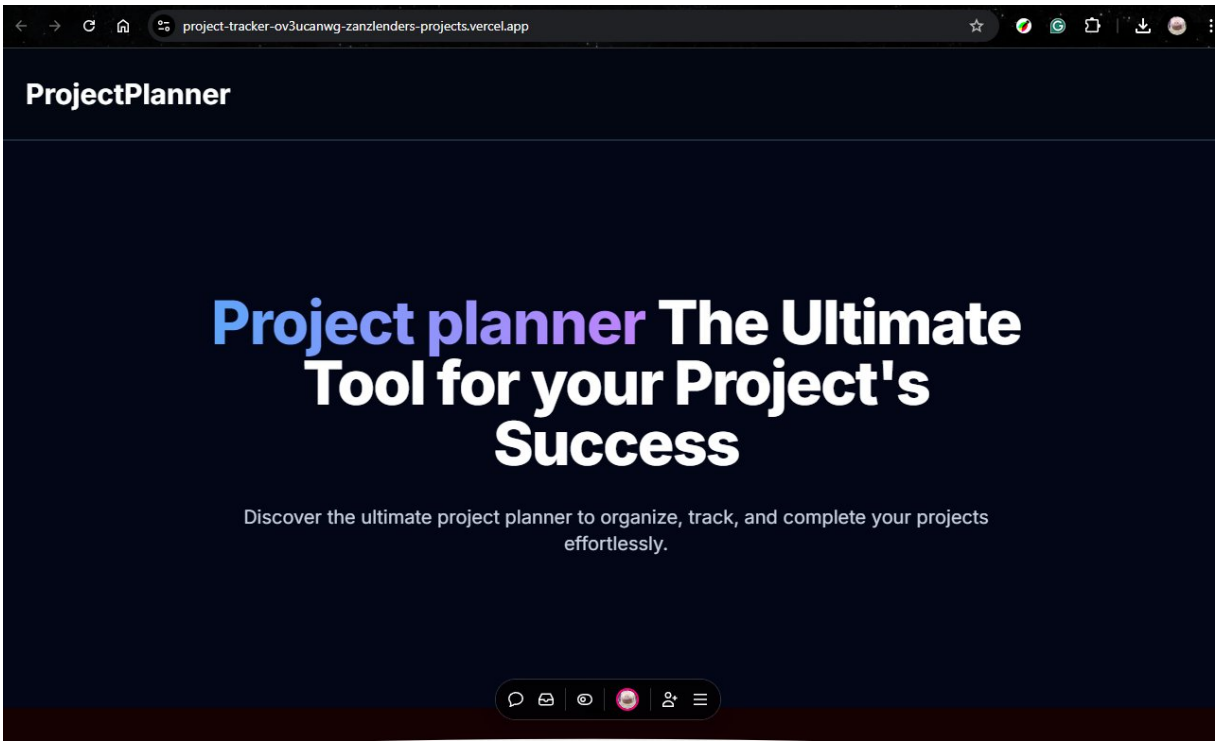
Slika 6. Trenutna početna stranica stranica Project plannera (Izvor: Vlastita izrada)

U jednom od klasičnih pristupa razvoja aplikacija, Github flowu, programer koji želi napraviti promjenu kreira novu granu na repozitoriju, neka bude nazvana „feat-change-landing-text“. Promjene se spreme na sustavu za verzioniranje (eng. *push*) i kreira se zahtjev za spajanje (eng. *pull request*) [70]. Vercel koji je integriran sa Github projektom automatski detektira promjene i ažurira sam zahtjev, kao što je vidljivo na slici 7.



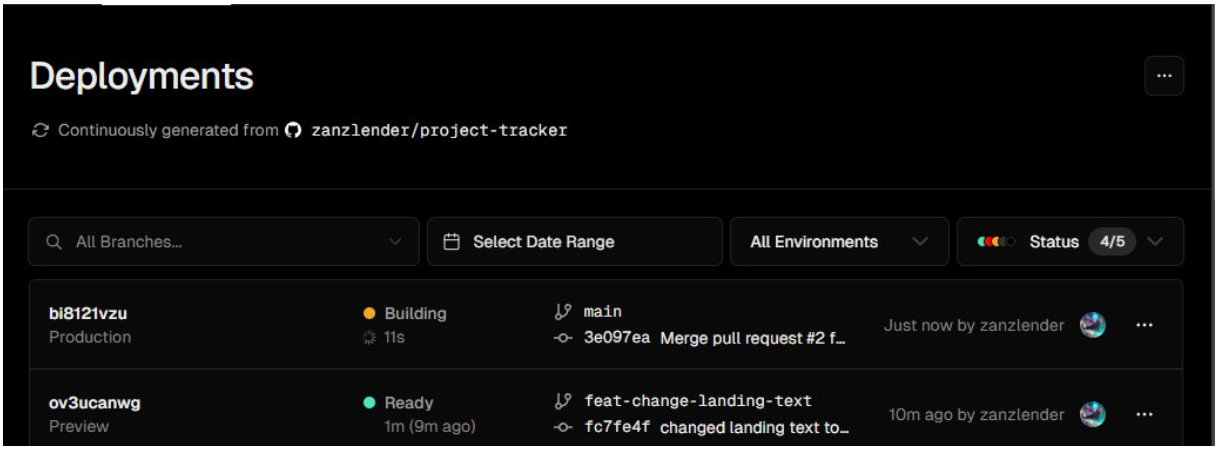
Slika 7. Zahtjev za promjenama (Izvor: Vlastita izrada)

Automatizirano se provodi proces verifikacije koda te se mogućnost spajanja zahtjeva na glavnu granu onemogućuje dok svi testovi nisu prošli. Tada se omogućuje jednostavno spajanje klikom na gumb „Merge pull request“. Također, valja istaknuti mogućnost za pretpregled koja se otvara klikom na gumb „View deployment“, što je vidljivo na narednoj slici.



Slika 8. Pretpregled promjena (Izvor: Vlastita izrada)

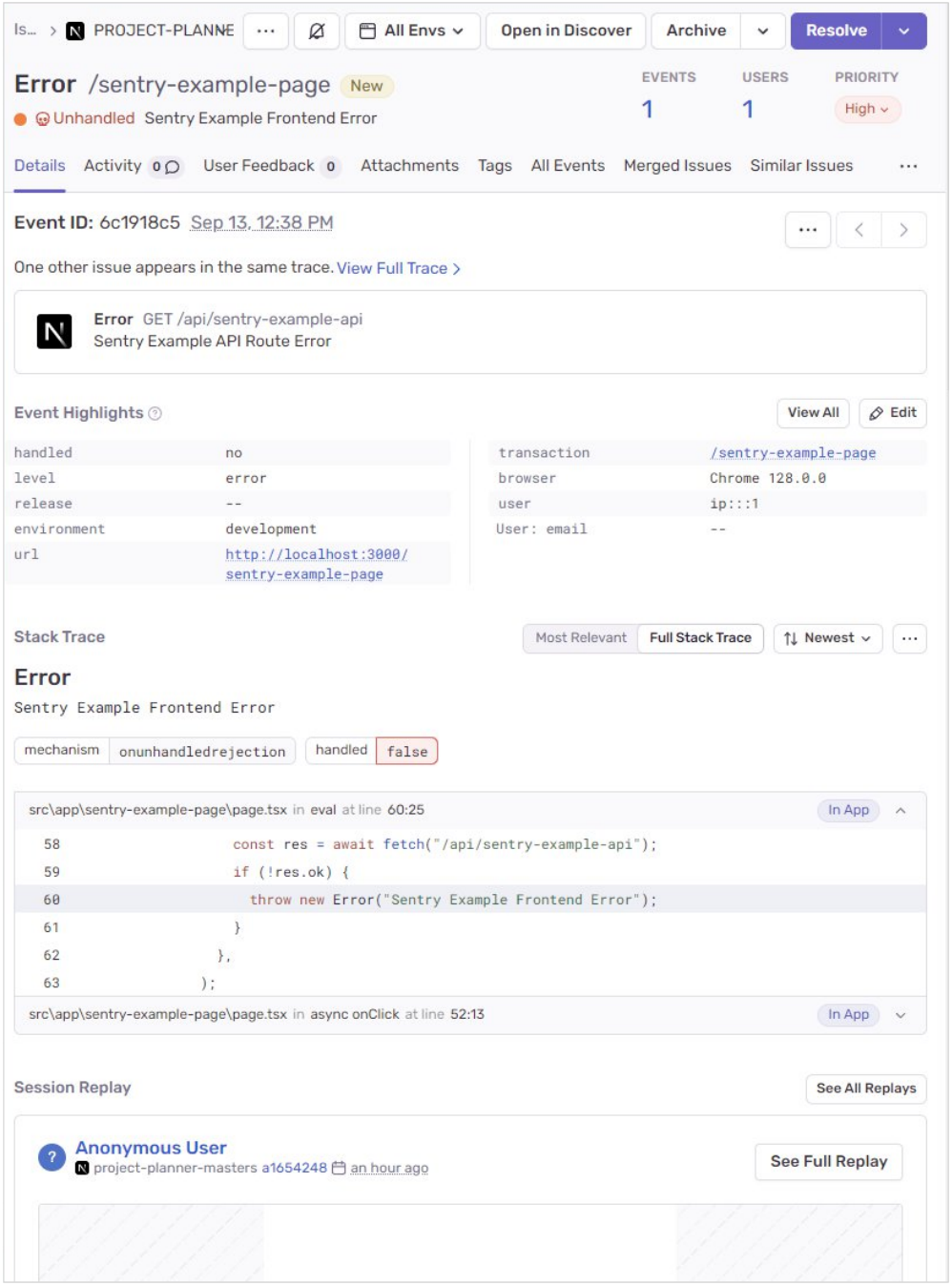
Valja još zamijetiti putanju stranice, Vercel je generirao novu putanju isključivo za ovu promjenu i moguće je provjeriti je li sve ispravno na Vercolovoj infrastrukturi. Također, na alatnoj traci pri dnu postoji mogućnost za postavljanje komentara od strane drugih programera za tu specifičnu promjenu. Jednom kada su promjene zadovoljavajuće, klikom na gumb za spajanje promjena pokreće se proces posluživanja aplikacije, što je uočljivo na slici 9.



Slika 9. Automatizirano pokretanje posluživanja na Vercelu (Izvor: Vlastita izrada)

Jednom kada je aplikacija uspješno izgrađena, poslužena je na postavljenoj glavnoj domeni, što je u slučaju aplikacije Project planner https://projectplanner.online.

U kontekstu nadogradnje aplikacije valja spomenuti i praćenja aplikacije, kako spremljeni podaci daju korisne uvide u potencijalne probleme u radu aplikacije. Servis Sentry ugrađen je u samu aplikaciju te može presretati sve greške koje se dogode unutar aplikacije, može pratiti točan trenutak kada se greška dogodila i koji koraci od strane korisnika su izazvali grešku. Uz sve navedeno, moguće je postaviti upozorenja gdje će se u određenom slučaju, primjerice ako se desi mnogo grešaka odjednom, poslati poruka zaduženim osobama. Na narednoj slici vidljiv je izvještaj jedne takve greške.



Slika 9. Prikaz greške unutar aplikacija u sklopu servisa Sentry (Izvor: Vlastita izrada)

Kao što se dade uočiti na slici, svaka greška koja se desi unutar sustava je zabilježena. Prikazane su informacije o razvojnoj okolini, točna linija koda u samoj aplikacija gdje se greška dogodila, informacije o korisniku te čak snimka sesije tog korisnika.

Primjeri definirani u ovome poglavlju nastoje prikazati jednostavnost nadogradnje aplikacije pomoću odabranih tehnologija i servisa. Proces CI/CD-a usko je povezan sa radom programera, no ne sputava ga, već mu pomaže da čim jednostavnije i kvalitetnije odradi promjene. Od trenutka kada programer pohrani promjene na sustavu za verzioniranje u manje od minutu vremena vidljivo je da li aplikacija radi kako treba, uključujući pretpregled promjena, a potvrđenim promjenama treba samo jednako toliko da budu poslužene kao najnovija verzija aplikacije svim korisnicima. Jednako tako, dostupne su sve informacije potrebne programeru kako bi mogao kvaliteno unaprijediti, kao što su metrike korištenja ili greške samog sustava.

9.4. Utjecaj korištenja smjernice za razvoj web aplikacije

Razvoj kvalitetne moderne web aplikacije, njeno unaprjeđivanje i održavanje zahtijevaju odabir ispravnih tehnologija i servisa. S tim ciljem te kako bi pojednostavile proces, za razvoj aplikacije Project planner korištene su smjernice definirane u ovome radu.

Vodeći se smjernicama definiran je čitav projekt i tehnološki stog za nj. Krenuvši od osnovnog koncepta projekta, ideje, formiraju se funkcionalni i tehnološki zahtjevi projekta koji su neophodni za odluku o odabiru tehnologija, alata i servisa. Nadalje, tehnologije i servisi koji su proizašli iz smjernicama slijede ideju „gradite sigurnosne mreže, ne ograde“ koju je definirao Browne [62]. Ova analogija razmatrana u kontekstu razvoja web aplikacija i specifičnije, postupanje sa pogreškama u razvoju, nastoji pojasniti kako je nemoguće izbjeći sve greške, jer greške u kodu uvijek će se desiti, ali bitno je kako ih se rješava. Stoga nije najbitnije graditi ograde, primjeice rigorozno provoditi testove pri razvoju, već imati definirane procese, sigurnosne mreže, kako ih riješiti. U prethodnom poglavlju bila je prikazana sigurnosna mreža u pogledu sustava za praćenje i bilježenje te kako je jednostavno napraviti izmjene u samoj aplikaciji koristeći informacije dobivene iz njih. Sve to omogućeno je kvalitetnim odabirom CI/CD lanca, integracijom sa poslužiteljem aplikacije te kvalitetnim tehnologijama servisima koji su bili definirani ograničenjima smjernica. Skupno razmatrano, čine iskustvo programiranja kvalitetnijim, iako ne pojednostavljuju samu implementaciju, kako ona ovisi o znanju programera.

Valjda napomenuti da iako sve spomenute prednosti smjernica samostalno mogu biti postignute iskustvom programera, vrijedi ih imati definirane radi boljeg svhaćanja i definicije problematike projekta. Samim time smanjuje se mogućnost pogrešnog odabira ili zaboravljanja na neke aspekte razvoja web aplikacija.

10. Zaključak

Razvoj web aplikacija široka je domena prožeta raznim mišljenjima i izvorima informacija koji često daju kontradiktorna ili neadekvatna rješenja. Cilj ovoga rada bio je analizirati trenutne metodologije razvoja web aplikacija generalno i specifčno sa TypeScriptom te definirati smjernice koje će programerima olakšati donošenje odluka pri odabiru kvalitetnih tehnologija za razvoj. Analiziran je TypeScript kao najkorišteniji skripti jezik na webu, uključujući razloge zašto je idealan razvoj web aplikacija, kao i razlozi kada ga valja izbjegavati. Istražene su trenutne agilne metodologije sa naglaskom na razvoj web aplikacija i integriranost u životni ciklus razvoja web aplikacija. Zaključeno je kako su sve istražene metodologije pregenralizirane kako bi najbolje ususstavile razvoj web aplikacija sa TypeScriptom te postoji mogućnost za unaprjeđenje istih. Smjernice navedene u ovome radu najviše su fokusirane na ubrzan razvoj aplikacije, skalabilnost i dobro programersko iskustvo. Provedena je empirijska analiza najboljih praksi u TypeScript ekosustavu i industriji Rezultatom analize definirano je 11 smjernica koje obuhvaćaju specifične korake u razvoju web aplikacija. Unutar smjernica analizirane su i argumentirane razne tehnologije i servisi te su odabrani one na presjeku kvalitete, sigurnosti, jednostavnosti korištenja, pouzdanosti i kvaliteti integracije sa TypeScriptom. Prilikom odabira naglasak je stavljen na modularnost, gdje svaka tehnologija relativno jednostavno može biti zamijenjena, ipak, veliko odvajanje od preporučenih tehnologija može dovesti do raznih integracijskih problema koje je tada potrebno vlastoručno riješiti. Finalno, same smjernice provedene su na praktičnom primjeru izrade aplikacije za praćenje projekata, pri čemu je prikazan utjecaj smjernica na proces razvoja, nadogradnje i održavanja aplikacije.

Prema svim istraživanjima spomenutim unutar ovog rada kao i zadovoljavjućim rezultatom pri praktičnoj zaključuje se kako definirane smjernice čine dobru početnu točku za razvoj aplikacije. Ipak, u obzir trebaju biti uzeti zahtjevi aplikacije, jer smjernice možda neće moći pokriti funkcionalne zahtjeve svih mogućih aplikacija.

Popis literature

[1] *System Architecture.* (bez dat.) U SeboWiki. Preuzeto 13.01.2024. s https://sebokwiki.org/wiki/System_Architecture

[2] Computer Science.org (2023.) *frontend vs. backend: What's the Difference?.* Preuzeto 13.01.2024. s https://www.computerscience.org/bootcamps/resources/frontend-vs-backend/

[3] GitLab B.V (bez dat.) *CI/CD Explained.* Preuzeto 13.01.2024. s https://about.gitlab.com/topics/ci-cd/

[4] Državni zavod za statistiku [DZS] (2024.) *Prosječne mjesečne neto i bruto plaće zaposlenih u 2023.* Preuzeto 13.01.2024. s https://podaci.dzs.hr/hr/podaci/trziste-rada/place/

[5] GIT-SCM (bez dat.) *1.1. Getting Started - About Version Control.* Preuzeto 13.01.2024. s https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control

[6] Github inc. (bez dat.) *About Github.* Preuzeto 13.01.2024. s https://github.com/about

[7] Amazon Web Services Inc. [AWS] (bez dat.) *What is Web Hosting.* Preuzeto 13.01.2024. https://aws.amazon.com/what-is/web-hosting/

[8] WebsiteGroup (bez dat.) *A Detailed Guide to Different Types of Web Hosting.* Preuzeto 13.01.2024. s https://websitesetup.org/different-types-of-web-hosting/#VPS%20

[9] NameCheap Inc. (bez dat). *Shared hosting.* Preuzeto 13.01.2024. s https://www.namecheap.com/hosting/shared/

[10] hostinger.com (bez dat.) *Hostinger pricing.* Preuzeto 13.01.2024. s https://www.hostinger.com/pricing

[11] bluehost Inc. (bez dat.) *Bluehost.* Preuzeto 13.01.2024. s https://www.bluehost.com/

[12] Google LLC. (bez dat.) *What is a Virtual Private Server (VPS)?.* Preuzeto 13.01.2024 s https://cloud.google.com/learn/what-is-a-virtual-private-server

[13] DigitalOcean LLC. (bez dat.) *Droplets.* Preuzeto 13.01.2024. s https://www.digitalocean.com/pricing/droplets#basic-droplets

[14] Amazon Web Services Inc. [AWS] (bez dat.) *EC2.* Preuzeto 13.01.2024. s https://aws.amazon.com/ec2/

[15] Fly.io (bez dat.) *Fly.io Resource Pricing.* Preuzeto 13.01.2024. s https://fly.io/docs/about/pricing/

[16] *System design primer.* (bez dat.). U Github. Preuzeto 13.01.2024. s https://github.com/donnemartin/system-design-primer

[17] Browne, T. *STREAM VOD: T3 Env, Line of Prime, Toxic Engs and more.* [Video file]. Preuzeto 13.01.2024. s https://www.youtube.com/watch?v=7ECooGD6Jas

[18] Liquid Web LLC (2024). *Liquid Web Dedicated plans.* Preuzeto 13.01.2024. s https://go.liquidweb.com/dedicated-plans/

[19] A2 HOSTING (2024). *A2Hosting pricing.* Preuzeto 13.01.2024 s https://www.a2hosting.com/dedicated-server-hosting

[20] HostGator.com (2024). *HostGator pricing.* Preuzeto 13.01.2024. s https://www.hostgator.com/help/article/hosting-price-chart

[21] Vercel (2024). *Vercel pricing.* Preuzeto 13.01.2024. s https://vercel.com/pricing

[22] Salesforce.com (2024). *Heroku pricing.* Preuzeto 13.01.2024. s https://www.heroku.com/pricing

[23] Netlify (2024). *Netlify pricing.* Preuzeto 13.01.2024. s https://www.netlify.com/pricing/

[24] The Linux Foundation (2024). *Prometheus.* Preuzeto 13.01.2024. s https://prometheus.io/

[25] Grafana Labs (bez dat.) *Grafana.* Preuzeto 13.01.2024. s https://grafana.com/

[26] PostHog inc. (bez dat.) *PostHog.* Preuzeto 13.01.2024.s https://posthog.com/

[27] Plausible Analytics (bez dat.) *Plausible analytics.* Preuzeto 13.01.2024. s https://plausible.io/

[28] Umami Software inc. (bez dat.) *Umami.* Preuzeto 13.01.2024. s https://umami.is/

[29] Elasticsearch B.V. (bez dat.) *Elastic Stack.* Preuzeto 13.01.2024. s https://www.elastic.co/elastic-stack/

[30] *Winston* (2024). Preuzeto 14.01.2024. s https://github.com/winstonjs/winston

[31] *Pino documentation* (2024). Preuzeto 14.01.2024. s https://getpino.io/

[32] *tRPC.* (bez dat.). U Github. Preuzeto 13.01.2024. s https://github.com/trpc/trpc/blob/next/packages/client/src/createTRPCClient.ts

[33] Xu, A. (2021). System Design Interview: An Insider's Guide. Second Edition. Volume 1.

[34] Meta Open Source (2024). *React.* Preuzeto 14.02.2024. s https://react.dev/

[35] *Svelte.* (bez dat.) Preuzeto 14.02.2024. s https://svelte.dev/

[36] SolidJS (bez dat.) *SolidJS.* Preuzeto 14.02.2024. s https://www.solidjs.com/

[37] *Astro.* (bez dat.) Preuzeto 14.02.2024. s https://astro.build/

[38] You, E. (bez dat.) *VueJS.* Preuzeto 14.02.2024. s https://vuejs.org/

[39] Vercel (2024). *NextJS.* Preuzeto 14.02.2024. s https://nextjs.org/

[40] *SvelteKit.* (bez dat.) Preuzeto 14.02.2024. s https://kit.svelte.dev/

[41] *SolidStart.* (bez dat.) Preuzeto 14.02.2024. s https://start.solidjs.com/getting-started/what-is-solidstart

[42] *Remix.* (bez dat.) Preuzeto 14.02.2024. s https://remix.run/

[43] Nuxt (2024). *Nuxt.* Preuzeto 14.02.2024. s https://nuxt.com/

[44] Jenkins (2024). *Jenkins documentation*. Preuzeto 15.07.2024. s https://www.jenkins.io/

[45] Gitlab (2024). *Get started with GitLab CI/CD*. Preuzeto 15.07.2024. s https://docs.gitlab.com/ee/ci/

[46] Github (2024). *Github Actions*. Preuzeto 15.07.2024. s https://docs.github.com/en/actions

[47] Vercel (2024). *Deploying to Vercel*. Preuzeto 15.07.2024. s https://vercel.com/docs/deployments/overview

[48] OpenJS (2024). *ExpressJS*. Preuzeto 15.08.2024. s https://expressjs.com/

[49] OpenJS Foundation (2024). *Fastify*. Preuzeto 15.08.2024. s https://fastify.dev

[50] Kamil Mysliwiec (2024). *NestJS*. Preuzeto 15.08.2024. s https://nestjs.com

[51] StackOverflow (2024). *2024 Developer survey – Technologies.* Preuzeto 15.08.2024. s https://survey.stackoverflow.co/2024

[52] David Watson (2024). *JavaScript and TypeScript Trends 2024: Insights From the Developer Ecosystem Survey.* Preuzeto 23.08.2024. s https://blog.jetbrains.com/webstorm/2024/02/js-and-ts-trends-2024/

[53] MDN (2024). *JavaScript.* Preuzeto 23.08.2024. s https://developer.mozilla.org/en-US/docs/Web/JavaScript

[54] Microsoft (2024). *Why create TypeScript?* Preuzeto 23.08.2024. s https://www.typescriptlang.org/why-create-typescript/

[55] CodeWorks (2024). *What is typescript and why you should use it.* Preuzeto 23.08.2024. s https://codeworks.me/blog/what-is-typescript-and-why-you-should-use-it/

[56] Matteo Possamai (2023). *Golang VS TypeScript: which one should you choose.* Preuzeto 23.08.2024. s https://medium.com/codex/golang-vs-typescript-which-one-should-you-choose-59132bc9a35b

[57] Janes, A. A., & Succi, G. (2012, October). *The dark side of agile software development. In Proceedings of the ACM international symposium on New ideas, new paradigms, and reflections on programming and software* (pp. 215-228)

[58] Robinson, E. (2005). Software Development Practices: The Good, the Bad and the Ugly. Journal of Advancing Technology, 2, 46-59.

[59] Gapuenko, J. (2023). *Web development methodologies and approaches.* Preuzeto 02.09.2024. s https://www.adcisolutions.com/knowledge/web-development-methodologies-and-approaches

[60] Statista (2022). *Breakdown of software development methodologies practiced worldwide in 2022.* Preuzeto 02.09.2024. s https://www.statista.com/statistics/1233917/software-development-methodologies-practiced/

[61] Meyer, B. (2014). Agile!: The good, the hype and the ugly. Springer Science & Business Media.

[62] Browne, T. *I interviewed Uncle Bob.* [Video file]. Preuzeto 02.09.2024. s https://www.youtube.com/watch?v=UBXXw2JSloo

[63] Kanban University (2022). *State of Kanban Report 2022.* Preuzeto 02.09.2024. s https://kanban.university/state-of-kanban/

[64] Attlasian (2024). *4 Kanban Principles for Agile Project Management.* Preuzeto 02.09.2024. s https://www.atlassian.com/agile/project-management/kanban-principles

[65] Chauhan, A., & Misra, R. (2023). *Outline of Web Development Life cycle in Software Engineering*. International National Conference on Recent Trends in Engineering & Technology

[66] Supabase inc. (2024). *Supabase.* Preuzeto 04.09.2024. s https://supabase.com/

[67] Collins, I. (2024). *NextAuth.js*. Preuzeto 06.09.2024. s https://https://next-auth.js.org/

[68] Lucia Auth (2024). *Lucia documentation.* Preuzeto 06.09.2024. s https://lucia-auth.com/

[69] Clerk (2024). *Clerk.* Preuzeto 06.09.2024. s https://clerk.som

[70] Github Inc. (2024). *Github Flow.* Preuzeto 06.09.2024. s https://docs.github.com/en/get-started/using-github/github-flow

Popis slika

Slika 1. Pojednostavljeni primjer širokog pogleda arhitekture sustava (Izvor: Vlastita izrada) 15
Slika 2. Vizualizacija ELK stack-a (Izvor: [ELK], preuzeto: 08.02.2024.).....35
Slika 3. “Line of prime” (Izvor: Vlastita izrada prema: [17]).....43
Slika 4. Prikaz smjernica za razvoj aplikacije u obliku grafa odlučivanja (Izvor: Vlastita izrada) 50
Slika 5. Visoka arhitektura aplikacije Project planner (Izvor: Vlastita izrada).....55
Slika 6. Početna stranica aplikacije Project planner (Izvor: Vlastita izrada).....57
Slika 7. Stranica za prijavu aplikacije Project planner (Izvor: Vlastita izrada).....57
Slika 8. Prikaz unosa korisničko imena aplikacije Project planner (Izvor: Vlastita izrada).....58
Slika 9. Stranica za kreiranje projekta aplikacije Project planner (Izvor: Vlastita izrada).....59
Slika 10. Prikaz svih projekata korisnika aplikacije Project planner (Izvor: Vlastita izrada)....59
Slika 11. Stranica za upravljanje članovima aplikacije Project planner (Izvor: Vlastita izrada) 60
Slika 12. Prikaz poziva u projekt aplikacije Project planner (Izvor: Vlastita izrada).....60
Slika 13. Stranica zadataka aplikacije Project planner (Izvor: Vlastita izrada).....61
Slika 14. Dodavanje novog zadatka aplikacije Project planner (Izvor: Vlastita izrada).....61
Slika 15. Sortiranje zadataka aplikacije Project planner (Izvor: Vlastita izrada).....62
Slika 16. Postavke projekta aplikacije Project planner (Izvor: Vlastita izrada).....62
Slika 6. Trenutna početna stranica stranica Project plannera (Izvor: Vlastita izrada).....63
Slika 7. Zahtjev za promjenama (Izvor: Vlastita izrada).....64
Slika 8. Pretpregled promjena (Izvor: Vlastita izrada).....65
Slika 9. Automatizirano pokretanje posluživanja na Vercelu (Izvor: Vlastita izrada).....65
Slika 9. Prikaz greške unutar aplikacija u sklopu servisa Sentry (Izvor: Vlastita izrada).....66

Popis tablica

Tablica 1. Primjer definiranja funkcionalnih zahtjeva 14
Tablica 2. Usporedba frontend biblioteka 20
Tablica 4. Prednosti i nedostaci Githuba za verzioniranje koda 25
Tablica 5. Prednosti i nedostaci Gitlaba za verzioniranje koda 26
Tablica 6. Prednosti i nedostaci BitBucketa za verzioniranje koda 27
Tablica 7. Usporedba servisa za dijeljeno posluživanje (Izvor: Vlastita izrada prema [9], [10], [11]) 28
Tablica 8. Usporedba servisa za VPS posluživanje (Izvor: Vlastita izrada prema [13], [14], [15]) 30
Tablica 9. Usporedba servisa za dedicirano posluživanje (Izvor: Vlastita izrada prema [18], [19], [20]) 31
Tablica 10. Usporedba servisa za posluživanje u oblaku (Izvor: Vlastita izrada prema [21], [22], [23]) 32
Tablica 11. Usporedba alata za praćenje i bilježenje (Izvor: Vlastita izrada prema [25], [26], [27], [28], [29]) 36
Tablica 12. Prednosti i nedostaci Jenkinsa (Izvor: Vlastita izrada prema [44]) 39
Tablica 13. Prednosti i nedostaci Gitlab CI/CD-ja (Izvor: Vlastita izrada prema [45]) 40
Tablica 14. Prednosti i nedostaci Github Actionsa (Izvor: Vlastita izrada prema [46]) 41
Tablica 15. Prednosti i nedostaci Vercela (Izvor: Vlastita izrada prema [47]) 42
Tablica 15. Smjernice za izradu web aplikacija 46
Tablica 16. Skraćene smjernice za izradu web aplikacija 47
Tablica 17. Primjer definiranja funkcionalnih zahtjeva 52
Tablica 18. Primjena smjernica za izradu web aplikacija za aplikaciju „Project planner“ 55