Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004:007
DOI: https://doi.org/10.24867/25OI03Kozma

STATIČKA ANALIZA KODA VEB APLIKACIJE ZA PODRŠKU POSLOVANJA ELEKTRONSKE PRODAVNICE KNJIGA PRIMENOM ALATA *SONARQUBE*

SONARQUBE-BASED STATIC CODE ANALYSIS OF AN ONLINE BOOKSTORE WEB APPLICATION

Nina Kozma, *Fakultet tehničkih nauka, Novi Sad*

Oblast – INŽENJERSTVO INFORMACIONIH SISTEMA

Kratak sadržaj – *Ovaj rad daje prikaz rezultata sprovedene statičke analize koda nad veb aplikacijom koja je razvijena u cilju obezbeđivanja podrške poslovanju elektronske prodavnice knjiga. Statička analiza koda je izvršena primenom alata SonarQube, dok je sama veb aplikacija implementirana pomoću MERN softverskog paketa. Rezultati statičke analize koda sistematično su predstavljeni kroz ovaj rad, te je priložen opis rešenja svih detektovanih problema u kodu. Na kraju rada, izvedeni su zaključci o izvršenoj analizi izvornog koda pomoću alata SonarQube, te su predloženi dalji pravci razvoja, održavanja i nadogradnje ovog informacionog sistema.*

Ključne reči: *Veb aplikacija, statička analiza koda, SonarQube, MERN stek*

Abstract – *This paper represents the results of a conducted SonarQube-based static code analysis of an online bookstore web application. The static code analysis was performed using SonarQube, while the web application was developed using the MERN technology stack. The results of the static code analysis are systematically presented and a a description of a potential solution for each problem detected in the source code is presented. is also attached. Finally, conclusions were drawn on the topic of conducted static code analysis using SonarQube, and further directions for the development, maintenance, and upgrading of this information system were proposed.*

Keywords: *Web Application, Static Code Analysis, SonarQube, MERN stack*

1. UVOD

Statička analiza izvornog koda igra značajnu ulogu u okviru procesa razvoja softverskog proizvoda, jer omogućava blagovremeno identifikovanje i otklanjanje postojećih grešaka i kritičnih delova u izvornom kodu aplikacije. Na taj način, programeri imaju mogućnost eliminacije ovih problema, pre nego što se softverski proizvod isporuči krajnjem korisniku. S tim u vezi, predmet izrade ovog rada predstavlja sprovođenje statičke analize koda veb aplikacije, primenom alata *SonarQube*.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Darko Stefanović, red. prof.

Veb aplikacija je razvijena u cilju unapređenja podrške poslovanja elektronske prodavnice knjiga, kao deo diplomskog rada u toku trajanja osnovnih akademskih studija na Fakultetu tehničkih nauka, u okviru studijskog programa – Inženjerstvo informacionih sistema [1], primenom *MERN* softverskog paketa, pri čemu *MERN* označava akronim za *MongoDB* bazu podataka, *Express* modul, *React* biblioteku i *Node.JS* platformu, respektivno. Ovaj rad podeljen je na četiri poglavlja. Naredno poglavlje pruža uvid u metodologiju rada, odnosno daje opis arhitekture razvijene veb aplikacije, kao i značaja sprovođenja statičke analize koda. Treće poglavlje prikazuje detektovane greške u kodu od strane alata za statičku analizu – *SonarQube*, dok četvrto poglavlje opisuje moguće načine njihovog otklanjanja. Poslednje poglavlje izvodi zaključke sprovedenog istraživanja i nudi smernice za dalji rad.

2. METODOLOGIJA RADA

Implementirani informacioni sistem predstavlja primer sistema za podršku planiranju poslovnih resursa (eng. *Enterprise Resource Planning*), jer omogućava efikasno planiranje svih resursa poslovanja, kao i praćenje procesa rada elektronske prodavnice knjiga. U cilju ostvarivanja održive konkurentske prednosti na tržištu, kao i radi unapređenja efikasnosti poslovanja i postizanja uspeha, savremene kompanije su u obavezi da sprovode kontinuiranu procenu rada uvedenih informacionih sistema, te da vrše njihovu redovnu optimizaciju i dalju nadogradnju. S tim u vezi, od izrazitog je značaja blagovremeno sprovođenje statičke analize koda, u cilju detekcije postojećih grešaka i sigurnosnih propusta u izvornom kodu aplikacije. Pored toga, statička analiza koda se primenjuje i radi unapređenja kvaliteta softvera, čitljivosti programskog koda, performansi izvršavanja aplikacije, te postizanja usaglašenosti sa definisanim standardima i najboljim praksama programiranja [2].

2.1. Opis arhitekture aplikacije

Kako bi se omogućila kontinuirana i konzistentna podrška poslovanju, implementirane su sledeće funkcionalnosti u okviru veb aplikacije: korisnička registracija i prijava na platformu, elektronsko poručivanje proizvoda, elektronsko plaćanje kreiranih porudžbina pomoću platforme *Paypal*, te unos korisničkih recenzija na odabrane knjige. Dodatno, aplikacija obezbeđuje kreiranje i izmenu evidencija o korisnicima aplikacije, nastalim porudžbinama, kao i proizvodima koji čine ponudu elektronske prodavnice.

Arhitektura kreirane *MERN* veb aplikacije sastoji se iz sledećih slojeva:

1. Sloj podataka čini baza podataka *MongoDB*,
2. Sloj poslovne logike – serverska aplikacija, razvijena pomoću platforme *Node.JS* i modula *Express* i
3. Sloj korisničkog interfejsa – klijentska aplikacija, napravljena uz pomoć *React* biblioteke.

MERN softverski paket predstavlja sve popularniji stek tehnologija koji se koristi za razvoj savremenih veb aplikacija. Pri tome, programeri koriste samo jedan programski jezik – *JavaScript*, što u velikoj meri ubrzava i olakšava proces razvoja aplikacije. Dodatno, sve komponente *MERN* steka su otvorenog koda, što smanjuje troškove razvoja aplikacije. Osim toga, primena *MERN* steka obezbeđuje skalabilnost i fleksibilnost u arhitekturi aplikacije, te se ona može prilagoditi specifičnim potrebama korisnika, ukoliko za time postoji potreba [3].

2.2. Statička analiza koda

Statička analiza koda podrazumeva kolekciju algoritama i tehnika koji se koriste za analizu izvornog koda, u cilju pronalaženja postojećih i potencijalnih grešaka i propusta u kodu, kao i radi detekcije loše prakse kodiranja. Pri tome, sama statička analiza se vrši bez izvršavanja programskog koda i takođe se smatra jednim od načina za automatizaciju procesa pregleda koda [4].

Prvobitna svrha statičke analize koda je bila optimizacija rada kompajlera, što je u međuvremenom prošireno dodatnim funkcionalnostima, poput otkrivanja nedostataka i grešaka u kodu. Shodno tome, u sve većoj meri se razvijaju gotovi softverski alati koji predstavljaju podršku programerima u sprovođenju statičke analize koda [5].

Alati za statičku analizu pretražuju izvorni kod aplikacije za specifičnim setom obrazaca ili pravila. Pojedini alati čak omogućavaju ručno definisanje novih pravila [5]. Po završetku analize, alat generiše izveštaj sa rezultatima, te ukazuje na određena odstupanja od propisanih standarda kvaliteta koda. Pored toga, alati za statičku analizu koda navode i uzrok određenog defekta, kao i način njegovog otklanjanja [2]. Bitno je naglasiti da odluku o izmeni koda, odnosno eliminisanje greške, izvršava sam programer.

2.3. Alat *SonarQube*

SonarQube predstavlja automatizovani alat za kontinuirani pregled koda, u cilju detekcije grešaka u izvornom kodu, kritičnih tačaka, kao i potencijalno rizičnih delova u kodu [6]. Napisan je u Java programskom jeziku, ali obezbeđuje podršku za 27 različitih programskih jezika, kao što su *C, C++, Java, JavaScript, PHP, GO, Python*, itd [7].

SonarQube omogućava programerima da pišu tzv. „čistiji“ kod, odnosno jednostavniji i pregledniji kod koji je usklađen sa dobrim praksama programiranja. Na osnovu rezultata statičke analize koda primenom alata *SonarQube,* obezbeđen je pregled postojećih grešaka u kodu, a koje mogu da dovedu do nepredviđenog ponašanja aplikacije, kao i pregled kritičnih tačaka programa koje mogu ugroziti rad aplikacije. Time se značajno smanjuje kompleksnost izvornog koda, otklanjanju se rizične tačke i stiču pogodni uslovi za optimizaciju rada aplikacije. Drugim rečima, primenom alata *SonarQube*, programeri imaju redovan uvid u kvalitet napisanog koda, što dalje doprinosi unapređivanju njihovih tehnika i načina kodiranja [8].

SonarQube obuhvata 3 domena, i to: domen održivosti – *code smell*-ovi, domen pouzdanosti – greške u kodu, te bezbednosni domen – ranjivosti i bezbednosno kritične tačke [9].

3. PRIKAZ DETEKTOVANIH PROBLEMA

Izrada rada podrazumevala je učitavanje odgovarajućih projekata u alat *SonarQube*, i to: serverske strane (eng. *Backend*) i klijentske strane (eng. *Frontend*) veb aplikacije namenjene za podršku poslovanja elektronske prodavnice knjiga.

Prvobitna analiza *backend* dela aplikacije je detektovala postojanje 21 *code smell*-a i jedne sigurnosno kritične tačke.

Najveći broj *code smell*-ova manjeg nivoa ozbiljnosti je prouzrokovan neispravnim imenovanjem fajlova u okviru *backend* projekta. Po konvenciji imenovanja, naziv promenljive koja se *export*-uje u okviru fajla mora da se poklapa sa nazivom samog fajla u okviru kog se ta promenljiva nalazi. Obezbeđivanjem pravilnog imenovanja fajlova, povećava se čitljivost koda i olakšava njegovo održavanje.

Sledeći *code smell* većeg nivoa ozbiljnosti na serverskoj strani aplikacije tiče se deklarisanja promenljive u spoljašnjem opsegu koju potencijalno može da „sakrije“ (eng. *Override*) druga promenljiva u unutrašnjem opsegu. Pri tome, obe promenljive nose isti naziv. U saglasnosti sa *SonarQube*-om, ukoliko je promenljiva koja je deklarisana u spoljašnjem opsegu „sakrivena“ od strane neke druge promenljive u nastavku koda, time se značajno pogoršava čitljivost koda, a samim tim i održavanje ovog dela aplikacije. Takođe, ovo može da dovede do zabune prilikom pregleda koda, odnosno, do nejasnoće koja promenljiva se tačno koristi u datom trenutku.

Sigurnosno kritična tačka na *backend*-u se odnosi na upotrebu argumenata komandne linije u izvornom kodu, što se smatra rizičnim potezom, te je pre njihove upotrebe u kodu, neophodno izvršiti validaciju, u cilju povećanja sigurnosti koda i smanjenja rizika od zloupotrebe sistema.

Sa druge strane, inicijalnom analizom *frontend* dela aplikacije, detektovano je 37 *code smell*-ova, tri ranjivosti, kao i jedna greška u kodu.

Najveći broj *code smell*-ova većeg nivoa ozbiljnosti na klijentskoj strani aplikacije tiče se upotrebe parametara koje imaju podrazumevajuću (eng. *Default*) vrednost. Saglasno *SonarQube* konvenciji, prilikom definisanja nove funkcije u kodu, parametri koji imaju podrazumevajuću vrednost trebalo bi da se nalaze iza parametara koji nemaju podrazumevajuću vrednost, što na ovom mestu nije ispoštovano.

Sledeći *code smell* na *frontend*-u većeg nivoa ozbiljnosti sličan je onom koji je detektovan na *backend*-u. Kao i na serverskoj strani aplikacije, i ovde postoji mogućnost „sakrivanja“ promenljive koja je deklarisana u spoljašnjem opsegu od strane druge promenljive deklarisane u unutrašnjem opsegu, te se preporučuje izmena ovog dela programskog koda.

Pronađene ranjivosti na *frontend*-u odnose se na upotrebu metoda *alert* i *confirm*, koje mogu da se koriste u toku procesa razvoja aplikacije, ali ne i nakon što je aplikacija isporučena krajnjim korisnicima, jer se time rizikuje izlaganje osetljivih informacija napadačima.

Greška višeg nivoa ozbiljnosti na *frontend*-u tiče se dva identična izraza čije se vrednosti porede pomoću operatora &&. Neophodno je promeniti jedan od izraza, kako bi se izbegla predvidljivost rezultata poređenja.

4. OPIS REŠENJA

Najveći broj *code smell*-ova manjeg nivoa ozbiljnosti serverske strane aplikacije je uzrokovan neadekvatnim dodeljivanjem naziva fajlu. Kako bi se otklonili spomenuti *code smell*-ovi, te postigao željeni nivo kvaliteta koda veb aplikacije, svi fajlovi su preimenovani na taj način, da promenljiva koja se *export*-uje u okviru fajla ima identičan naziv kao sam fajl. Time su ovi *code smell*-ovi uklonjeni.

Naredni *code smell* većeg nivoa ozbiljnosti na *backend*-u tiče se deklarisanja promenljive u spoljašnjem opsegu, koju potencijalno može da „sakrije“ druga promenljiva deklarisana u unutrašnjem opsegu. Shodno tome, doneta je odluka da se obe promenljive preimenuju, kako ne bi dolazilo do otežane čitljivosti i održavanja koda. Jasnim navođenjem naziva promenljivih, omogućeno je intuitivno zaključivanje njihovih uloga u izvornom kodu aplikacije, te je sam programski kod „čistiji“ i u skladu sa dobrim praksama programiranja.

Detektovana sigurnosno kritična tačka unutar izvornog koda serverske aplikacije odnosila se na upotrebu argumenata komandne linije. U skladu sa *SonarQube* konvencijom kodiranja, upotreba argumenata komandne linije se smatra rizičnim potezom. Usled toga, izvršena je validacija sigurnosti upotrebe sistema, te je ona pokazala da ne postoji sigurnosni rizik od zloupotrebe, s obzirom na to da se kroz argumente komandne linije ne prosleđuje nikakva osetljiva i poverljiva informacija.

Najveći broj *code smell*-ova većeg nivoa ozbiljnosti klijentske strane aplikacije tiče se upotrebe parametara koje imaju podrazumevajuću vrednost. Kao što je rečeno u prethodnom poglavlju, prilikom definisanja nove funkcije, neophodno je voditi računa o redosledu ulaznih parametara, i to na taj način, da se najpre definišu parametri koji nemaju podrazumevajuću vrednost, a potom parametri koji imaju *default*-nu vrednost. Međutim, ovde je reč o *Redux Reducer* funkciji.

Redux biblioteka omogućava kontrolu stanja i toka podataka *JavaScript* aplikacija, te se uglavnom koristi radi obezbeđivanja konzistentnog izvora podataka koji predstavlja trenutno stanje podataka aplikacije. Konkretno, u izvornom kodu se definiše tzv. *Reducer* funkcija, odnosno reduktor, koji prima ulazne parametre: stanje, koje reprezentuje trenutno stanje podataka aplikacije, te akciju koja vrši obradu ovih podataka. Na osnovu rezultata obrade, akcija vraća izmenjeno ili trenutno stanje [10].

U skladu sa zvaničnom dokumentacijom *Redux* tehnologije, potrebno je ispoštovati odgovarajuće konvencije koje se tiču pravilnog definisanja spomenutih *Reducer* funkcija. Saglasno *Redux* konvenciji za kreiranje reduktora, redosled definisanja ulaznih parametara podrazumeva da se na prvo mesto navodi parametar koji ima podrazumevajuću vrednost – stanje aplikacije, iza kojeg sledi sledi parametar koji nema podrazumevajuću vrednost – akcija [11]. Dakle, dolazi do određenih kontradiktornosti između *Redux* konvencije pisanja koda, sa jedne strane, i *SonarQube* pravila pisanja kvalitetnog programskog koda, sa druge strane. Iz tog razloga, kako bi se očuvala konzistentnost i čitljivost koda, svi *code smell*-ovi koji se tiču ovog problema označeni su kao lažno-pozitivni, te u narednoj analizi izvornog koda neće biti detektovani.

Ovim se takođe može uočiti da, uprkos značaju sprovođenja automatizovane analize izvornog koda primenom alata *SonarQube*, programeri igraju ključnu ulogu u okviru procesa izmene strukture aplikacije, te otklanjanje detektovanih grešaka u velikoj meri zavisi od odabranih tehnologija za razvoj softverskog proizvoda.

Sledeći *code smell* većeg nivoa ozbiljnosti na klijentskoj strani aplikacije naglašava mogućnost „sakrivanja“ promenljive koja je deklarisana u spoljašnjem opsegu od strane druge promenljive deklarisane u unutrašnjem opsegu. Ovaj *code smell* se javio na ukupno 5 mesta u izvornom kodu klijentske aplikacije.

Na prvom mestu u kodu, obe promenljive su nosile isti naziv, te je doneta odluka da se promenljiva deklarisana u unutrašnjem opsegu preimenuje, kako bi intuitivno bila jasna njena uloga u kodu. Na taj način, izbegnuto je otežano čitanje i održavanje koda, te detektovani *code smell* uspešno otklonjen.

Naredna četiri *code smell*-a bave se istim problemom. Reč je o promenljivama koje definišu inicijalno stanje aplikacije, u slučaju modifikacije podataka o proizvodu. Ukoliko administrator sistema nije uneo sve podatke o proizvodu prilikom ažuriranja evidencije, poput naziva autora knjige, pripadajućeg žanra, te izdavača i dobavljača knjige, klijentska strana aplikacije će dostaviti inicijalne vrednosti ovih obeležja serveru. U slučaju da je administrator ipak uneo nove vrednosti ovih obeležja u okviru odgovarajuće forme, one će biti isporučene serveru i kao takve, postavljene za nove vrednosti obeležja knjige unutar baze podataka. Shodno tome, može se zaključiti da u ovom slučaju, ne dolazi do sakrivanja promenljiva deklarisanih u različitim opsezima, već su ove promenljive neophodne za kontrolu toka podataka *JavaScript* aplikacije. Drugim rečima, nakon manuelne analize izvornog koda, može se uvideti da se radi o lažno-pozitivnom *code smell*-ovima, te su oni, kao takvi, i označeni.

Pronađene potencijalne ranjivosti u izvornom kodu klijentske aplikacije odnose se na metode *alert*(…) i *confirm*(…), koje ne bi trebalo da se koriste nakon što je aplikacija puštena u upotrebu, jer se time rizikuje izlaganje osetljivih informacija napadačima.

Konkretno, ukoliko je unos korisničke recenzije na knjigu bio uspešan, poziva se *alert* metoda, kao potvrda uspešnosti izvršavanja operacije. Kako je potvrđivanje uspešnosti operacije u ovom delu koda implementirano isključivo u okviru procesa detekcije grešaka od strane programera, linija koda koja poziva ovu metodu je obrisana, a pronađena ranjivost otklonjena.

Potom, naredne dve ranjivosti u kodu odnose se na upotrebu metode *confirm* prilikom brisanja proizvoda iz evidencije knjiga koje čine ponudu prodavnice, kao i u

slučaju brisanja korisničkog naloga iz evidencije korisnika. Od administratora sistema se prilikom izvršavanja spomenutih operacija zahteva potvrđivanje operacije brisanja, kako bi se izbegao scenario u kome dolazi do slučajnih grešaka, odnosno, slučajnih brisanja podataka iz sistema. Obe ranjivosti u kodu otklonjene su zamenom kritične *confirm* metode sa modalnim dijalogom, koji omogućava korisničku potvrdu brisanja proizvoda, odnosno korisnika iz evidencije.

Kada administrator sistema inicira proces brisanja proizvoda ili korisničkog naloga, otvara se modalni dijalog čijim se potvrđivanjem poziva odgovarajuća metoda brisanja. U slučaju da administrator zatvori modalni dijalog pre potvrđivanja operacije brisanja, ona neće biti izvršena, te se time sprečavaju slučajne greške uklanjanja podataka iz evidencije. Na taj način, otklonjene su obe ranjivosti u kodu, te je takođe obezbeđena njegova sigurna upotreba.

Greška u kodu višeg nivoa ozbiljnosti na *frontend*-u odnosi se na dva identična izraza čije se vrednosti porede pomoću operatora &&. Kako se u ovom slučaju poredi promenljiva sama sa sobom, te će rezultat poređenja uvek biti potvrdan, poređenje je obrisano, te je greška u kodu eliminisana.

Nakon što je pokrenuta ponovna analiza izvornog koda klijentske i serverske strane veb aplikacije za podršku poslovanja elektronske prodavnice knjiga pomoću alata *SonarQube*, sve greške u kodu, ranjivosti, sigurnosno kritične tačke, te *code smell*-ovi, uspešno su otklonjeni, dok je tehnički dug (eng. *Technical Debt*) sveden na nulu.

5. ZAKLJUČAK

Po završetku sprovođenja svih izmena u izvornom kodu aplikacije za podršku poslovanja elektronske prodavnice knjiga, napisani kod je u skladu sa *SonarQube* standardima i pravilima za pisanje kvalitetnog programskog koda, te su sve greške u kodu, potencijalne ranjivosti i sigurnosno kritične tačke, kao i *code smell*-ovi uspešno uklonjeni.

Može se zaključiti da je statička analiza koda pomoću alata *SonarQube* značajno unapredila kvalitet napisanog izvornog koda, te time doprinela i poboljšanju performansi rada aplikacije. Takođe, uz pomoć ovog alata, značajano je olakšano čitanje i tumačenje izvornog koda, te su veštine programera poboljšane kroz interakciju sa ovim alatom. Ipak, bitno je istaći da, iako primena alata za statičku analizu koda značajno ubrzava i olakšava proveru kvaliteta napisanog koda, sve odluke o promenama strukture koda donosi sam programer, te one u velikoj meri zavise od odabranih tehnologija za razvoj softverskog proizvoda.

Sprovođenjem statičke analize koda razvijene veb aplikacije, te njenim kontinuiranim održavanjem i daljom nadogradnjom, kompaniji je omogućeno da reaguje na dinamične uslove poslovnog tržišta, stekne uvid u performanse poslovanja kompanije i optimizuje upotrebu raspoloživih resursa.

Usled toga, može se zaključiti da savremene kompanije nastoje da upotrebe informacione sisteme poput spomenutog, kako bi optimizovale svoje poslovne procese, te ostvarile održivu konkurentnost.

5. LITERATURA

[1] N. Kozma, „Projektovanje informacionog sistema za podršku poslovanja prodavnice knjiga,“ Fakultet tehničkih nauka, Univerzitet u Novom Sadu, Novi Sad, Republika Srbija, 2021.

[2] M. Beller, R. Bholanath, S. McIntosh i A. Zaidman, „Analyzing the State of Static Analysis: A Large-Scale Evaluation in Open Source Software,“ u *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering*, Suita, Osaka, Japan, 2016.

[3] Y. Kadam, A. Goplani, S. Mattoo, S. K. Gupta, D. Amrutkar i J. Dhanke, „Introduction to MERN Stack & Comparison with Previous Technologies,“ *European Chemical Bulletin*, t. 12, br. 4, pp. 14382-14386, 2023.

[4] Ž. Aleksić, „Statička analiza koda zasnovana na upotrebi Roslyn kompajlera,“ *Zbornik radova Fakulteta tehničkih nauka,* t. 34, br. 01, 2018.

[5] D. Stefanović, D. Nikolić, D. Dakić, I. Spasojević i S. Ristić, „Static Code Analysis Tools: A Systematic Literature Review,“ U *31st Daaam International Symposium On Intelligent Manufacturing And Automation*, Vienna, Austria, 2020.

[6] „SonarQube Documentation,“ [На мрежи]. Available: https://docs.sonarqube.org/latest/. [Последњи приступ 22. 6. 2023.].

[7] „Languages - Overview,“ [Na mreži]. Available: https://docs.sonarqube.org/latest/analyzing-source-code/languages/overview/. [Poslednji pristup 22. 6. 2023.].

[8] K. Dissanayake, „SonarQube (Part 2) — Features of SonarQube, Installation and some practice on SonarQube,“ [Na mreži]. Available: https://medium.com/swlh/sonarqube-part-2-features-of-sonarqube-installation-and-some-practice-on-sonarqube-d523ae9a998a. [Poslednji pristup 22. 6. 2023.].

[9] „User Guide - Rules,“ [Na mreži]. Available: https://docs.sonarqube.org/latest/user-guide/rules/overview/. [Poslednji pristup 22. 6. 2023.].

[10] N. Subić, „Implementacija mobilne aplikacije Travelimg pomoću React Native radnog okvira,“ *Zbornik radova Fakulteta tehničkih nauka u Novom Sadu*, t. 34, p. 4, 2019.

[11] „Redux Fundamentals, Part 3: State, Actions, and Reducers,“ 23. 4. 2023.. [Na mreži]. Available: https://redux.js.org/tutorials/fundamentals/part-3-state-actions-reducers. [Poslednji pristup 8. 9. 2023.].

Kratka biografija:

Nina Kozma rođena je u Subotici 1998. godine. Student je master studija na Fakultetu tehničkih nauka, na studijskom programu – Inženjerstvo informacionih sistema
kontakt: nina.kozma@uns.ac.rs