Web-platforma za statističku obradu podataka s očuvanjem privatnosti

Grabar, Antonia

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Rijeka, Faculty of Engineering / Sveučilište u Rijeci, Tehnički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:190:385160

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Diplomski studij računarstva

Diplomski rad

Web-platforma za statističku obradu podataka s očuvanjem privatnosti

Rijeka, rujan 2024.

Antonia Grabar
0069088321

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Diplomski studij računarstva

Diplomski rad

Web-platforma za statističku obradu podataka s očuvanjem privatnosti

Mentor: prof.dr.sc. Kristijan Lenac

Rijeka, rujan 2024.

Antonia Grabar
0069088321

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 24.03.2024.

Zavod: Zavod za računarstvo
Predmet: Napredni operacijski sustavi

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Antonia Grabar (0069088321)**
Studij: Sveučilišni diplomski studij računarstva (1400)
Modul: Računalni sustavi (1442)

Zadatak: **Web-platforma za statističku obradu podataka s očuvanjem privatnosti / Web-platform for privacy-preserving statistical data processing**

Opis zadatka:

Dizajnirati i implementirati web platformu koja koristi Multi-Party Computation (MPC) protokole za očuvanje privatnosti pri agregaciji podataka. Sustav treba omogućiti izračunavanje agregatnih statistika osiguravajući da pojedinačni unosi svakog sudionika ostanu povjerljivi. Testirati funkcionalnost i sigurnost sustava simuliranjem scenarija s različitim vrstama i veličinama podataka.

Rad mora biti napisan prema Uputama za pisanja diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 20.03.2024.

Mentor:
prof. dr. sc. Kristijan Lenac

Predsjednik povjerenstva za diplomski ispit:
prof. dr. sc. Miroslav Joler

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2024.

_____
Ime Prezime

Zahvala

Zahvaljujem mentoru prof.dr.sc. Kristijanu Lencu na podršci i dragocjenim savjetima tijekom cijelog procesa pisanja ovog diplomskog rada.

Zahvaljujem obitelji i prijateljima na neizmjernoj podršci tijekom cijelog mog studijskog putovanja.

Zahvaljujem Riteh Web Team-u na prilici za stjecanje neprocjenjivog praktičnog znanja i iskustava koja ću pamtiti cijeli život.

Sadržaj

Popis slika

Poglavlje 1

Uvod

U današnjem digitalnom okruženju, privatnost podataka postala je jedan od najvažnijih izazova s kojima se društvo suočava. S razvojem tehnologije i sveprisutnom upotrebom interneta, količina osjetljivih podataka koja se svakodnevno razmjenjuje putem različitih platformi eksponencijalno raste. Bilo da se radi o osobnim informacijama, financijskim transakcijama ili zdravstvenim podacima, ovi podaci su često metom kibernetičkih napada. Kibernetičke prijetnje postaju sve složenije, a njihovi napadi sve precizniji, što stavlja dodatni pritisak na organizacije da pronađu i implementiraju učinkovite metode zaštite. Osim same krađe podataka, prijetnje se manifestiraju i kroz gubitak povjerenja korisnika u digitalne sustave, što može imati dugoročne posljedice na poslovanje i reputaciju organizacija.

Kako bi se nosili s ovim izazovima, nužno je primijeniti napredne tehnologije koje mogu osigurati zaštitu privatnosti na visokoj razini. U tom kontekstu, web-platforme koje omogućuju statističku obradu podataka moraju biti projektirane tako da zaštite korisničke informacije u svakom trenutku, bez obzira na opseg i vrstu podataka koji se obrađuju. To nije samo tehnički izazov, već i etička obveza prema korisnicima, čiji se podaci moraju tretirati s najvišom razinom povjerljivosti. Korištenje kriptografskih tehnika, poput sigurnog višestranačkog računanja (Secure Multiparty Computation (MPC)), postalo je ključno za postizanje ove razine zaštite. MPC omogućuje prikupljanje i analizu podataka na način koji štiti individualne informacije sudionika, osiguravajući da podaci ostanu privatni čak i tijekom složenih statističkih obrada. Ova tehnologija predstavlja budućnost zaštite podataka, pružajući organizacijama

alate potrebne za učinkovitu borbu protiv rastućih kibernetičkih prijetnji, a istovremeno omogućujući korisnicima sigurnost i povjerenje u digitalne procese.

1.1 Opis zadatka

Zadatak diplomskog rada bio je dizajnirati i implementirati web aplikaciju koja koristi MPC protokole za očuvanje privatnosti pri agregaciji podataka. Sustav treba omogućiti izračunavanje agregatnih statistika osiguravajući da pojedinačni unosi svakog sudionika ostanu povjerljivi. Budući da je područje primjene aplikacije ostalo na izbor, ova aplikacija realizirana je kao analiza sigurnosnih prijetnji tvrtki, prikupljajući podatke o učestalosti 10 različitih kibernetičkih napada tvrtki tijekom razdoblja od 12 mjeseci. Također, u ovom radu će se promotriti izazovi očuvanja privatnosti, zakonski okviri i regulative te analizirati MPC protokol.

Poglavlje 2

Očuvanje privatnosti

Očuvanje privatnosti ključno je u svim fazama obrade podataka, s naglaskom na privatnost ulaza i privatnost izlaza. Privatnost ulaza (engl. *input privacy*) osigurava da strana koja obrađuje podatke ne može pristupiti osobnim informacijama, međuvrijednostima ili statističkim rezultatima, osim ako su ti podaci izričito odabrani za dijeljenje.[1] Na primjer, u istraživačkom projektu koji koristi zdravstvene podatke, privatnost ulaza osigurava da istraživači ne mogu vidjeti osobne podatke pacijenata, osim u slučaju kada su ti podaci neophodni za provođenje analize. Ovo osigurava da su osobne informacije zaštićene od neovlaštenog pristupa i da se koriste samo na način koji je strogo potreban za istraživanje.

Privatnost izlaza (engl. *output privacy*) smanjuje rizik da se osobni podaci mogu zaključiti iz rezultata obrade, što je posebno važno kada se rezultati objavljuju ili dijele s većim brojem korisnika. Primjerice, kada se objavljuju podaci o prihodima u određenom gradu, privatnost izlaza osigurava da se ne mogu prepoznati specifične osobe na temelju tih informacija. Ovo znači da, iako su podaci dostupni za analizu ekonomskih trendova, pojedinačni identiteti ostaju zaštićeni i ne mogu se otkriti iz objavljenih statistika.

Osim privatnosti ulaza i izlaza, ključno je razmotriti i dugoročnu sigurnost podataka tijekom cijelog njihovog životnog ciklusa. Očuvanje privatnosti podataka zahtijeva neprekidno praćenje i prilagodbu sigurnosnih mjera. Na primjer, u slučaju korištenja podataka iz društvenih mreža za istraživanje, čak i kada su podaci anoni-

mizirani, postoji rizik da će napredne tehnike omogućiti povezivanje tih podataka s pojedincima. Zbog toga je važno ne samo implementirati privatnost ulaza i izlaza, već i osigurati da su podaci kontinuirano zaštićeni od novih prijetnji koje se mogu pojaviti s razvojem tehnologije. Održavanje transparentnosti prema korisnicima o tome kako se njihovi podaci obrađuju i koriste, te edukacija o važnosti postavki privatnosti također igraju ključnu ulogu u zaštiti osobnih podataka. Na taj način osigurava se da korisnici zadrže kontrolu nad svojim informacijama i da se podaci koriste na način koji poštuje njihovu privatnost.

2.1 Izazovi

S obzirom na sve veću prisutnost računalnih tehnologija, količina podataka proizvedenih svakodnevnim aktivnostima ljudi eksponencijalno raste, što stvara nove i neočekivane prijetnje privatnosti. Iako obrada tih podataka može donijeti korisne usluge, poput poboljšanja planiranja prometa u velikim gradovima, prikupljanje podataka o kretanju vozila može istovremeno omogućiti identifikaciju pojedinaca bez njihovog znanja. Kombinacija tih podataka s dodatnim informacijama, poput adresa stanovanja ili radnog mjesta, može dovesti do jedinstvene identifikacije osobe, čak i u velikom uzorku. Slično tome, društvene mreže predstavljaju značajan izazov za privatnost, budući da su dizajnirane za dijeljenje informacija o ljudima, ali njihovi stavovi i aktivnosti mogu se zloupotrijebiti. Iako se ulažu napori u zaštitu privatnosti kroz tehnološka rješenja, njihova stvarna učinkovitost ostaje neizvjesna zbog velike količine osobnih podataka, poput imena, fotografija i e-mail adresa, koje korisnici često dijele nesvjesno. Edukacija korisnika o postavkama privatnosti i potencijalnim rizicima ključno je za smanjenje kršenja privatnosti. Nadalje, računarstvo u oblaku dodatno komplicira očuvanje privatnosti jer vlasnici podataka gube kontrolu nad svojim informacijama, a pravna ograničenja su često nejasna zbog teškoće u određivanju gdje se podaci nalaze i obrađuju, što otežava primjenu zakonskih mjera zaštite.[2]

Izazovi očuvanja privatnosti u digitalnom dobu ne odnose se samo na sve veću količinu podataka, već i na složenost tehničkih, pravnih i organizacijskih prepreka koje se javljaju pri njihovoj obradi i zaštiti. Jedan od izazova je postizanje ravnoteže između privatnosti i korisnosti podataka. Naime, često se primjenjuju metode koje

dodaju šum ili ograničavaju dijeljenje podataka kako bi se očuvala privatnost, ali to može smanjiti točnost i korisnost analize. Pronalaženje optimalne ravnoteže između zaštite privatnosti i dobivanja kvalitetnih uvida iz podataka izuzetno je zahtjevno.

Također, primjena tehnika za očuvanje privatnosti u velikim razmjerima predstavlja značajan tehnički izazov, jer može biti računalno intenzivna, posebno kada se radi s velikim i složenim skupovima podataka. Ove tehnike često zahtijevaju značajnu količinu računalnih resursa i naprednu infrastrukturu, što može povećati troškove i složenost njihove implementacije. Na primjer, primjena enkripcije ili metoda diferencijalne privatnosti u velikim datasetovima može usporiti obradu podataka i zahtijevati dodatne računalne kapacitete za analizu i pohranu. Uz to, složenost naprednih tehnika zaštite privatnosti zahtijeva specijalizirane vještine i znanja koja nisu uvijek dostupna unutar organizacija, što dodatno otežava njihovu široku primjenu.[3]

Pored toga, organizacije moraju uskladiti svoje postupke s raznim regulatornim okvirima, što dodatno komplicira analizu velikih podataka uz očuvanje privatnosti.

2.2 Problem izračuna agregatnih statistika

Izračunavanje agregatnih statistika uz očuvanje privatnosti podataka, kao i pojedinaca, predstavlja složen izazov. Iako se agregatni podaci, poput prosjeka ili ukupnih zbrojeva, često koriste za razumijevanje šire slike i identificiranje trendova, oni ne nude potpunu sigurnost od otkrivanja osjetljivih informacija. Čak i kada su podaci naizgled anonimni, njihova kombinacija s dodatnim informacijama može omogućiti identifikaciju pojedinaca. Na primjer, prilikom izračuna prosječnih plaća zaposlenika u određenoj tvrtki, dodatni podaci poput radnog staža, radnih pozicija ili specifičnih sektora mogu, u kombinaciji s poznatim agregiranim podacima, dovesti do identifikacije konkretnih osoba, posebno u manjim timovima ili organizacijama gdje je broj zaposlenika ograničen.

Osim same zaštite privatnosti podataka, postoji i širi problem povjerenja u institucije koje prikupljaju i obrađuju ove podatke. Kada institucije računaju agregatne statistike, postoji rizik da će osjetljive informacije biti otkrivene ili zloupotrijebljene, što može imati ozbiljne posljedice za sigurnost i povjerenje pojedinaca. Primjerice,

institucije koje žele prikupiti podatke iz bolnica radi provođenja istraživanja i agregatne statistike, suočavaju se s ozbiljnim izazovom zaštite povjerljivosti tih podataka. Bolnice, koje čuvaju osjetljive informacije o pacijentima, često su sumnjičave prema dijeljenju podataka jer postoji rizik da će privatni podaci biti neovlašteno otkriveni ili zloupotrijebljeni. Iako su ovi podaci ključni za unapređenje zdravstvene skrbi i donošenje informiranih odluka, postoji stalna opasnost da se, čak i kroz anonimne ili deidentificirane podatke, mogu identificirati pojedinci, osobito u slučaju rijetkih bolesti ili specifičnih populacija. Osim toga, postoje i zakonski okviri koji strogo reguliraju način na koji se ti podaci mogu prikupljati, obrađivati i dijeliti, što dodatno otežava suradnju.

Uz opisane probleme nadovezuje se specifičan problem istraživanja frekvencija pojavljivanja kibernetičkih napada raznih tvrtki. Tvrtke koje žele doprinijeti analizi prijetnji često se suočavaju s dilemom između potrebe za detaljnim informacijama o napadima i očuvanja vlastite privatnosti. Da bi omogućile analizu, mnoge tvrtke koriste tehnike kao što su anonimizacija i agregacija podataka. Ove metode uklanjaju ili maskiraju specifične informacije koje bi mogle otkriti identitet tvrtke ili detalje o napadu. Iako anonimizacija i agregacija pomažu u skrivanju identiteta tvrtki, one nisu uvijek potpuno sigurne. Postoji rizik da se kroz kombinaciju anonimnih podataka ili upotrebom vanjskih informacija može reidentificirati izvor podataka. Na primjer, čak i kada se identitet tvrtke ukloni, specifične karakteristike napada mogu i dalje otkriti informacije koje omogućuju povratak do izvora. Također, ako se podaci previše generaliziraju, mogu pružiti netočnu sliku o učestalosti i vrsti napada. To može smanjiti točnost analize i otežati donošenje pravih odluka. Sve ove strategije mogu pružiti određeni stupanj zaštite, ali nijedna metoda nije savršena i može biti podložna sigurnosnim propustima.

2.3 Zakonski okviri i regulative

Prema članku 8. Povelje o temeljnim pravima Europske unije, svaki pojedinac ima sljedeća prava u vezi sa zaštitom osobnih podataka:[7]

- **Pravo na zaštitu osobnih podataka** - Osobni podaci mogu se prikupljati,

obrađivati i koristiti samo u skladu sa zakonom i na transparentan način

- **Pravo na pristup podacima** - Svaka osoba može pristupiti svojim podacima i tražiti njihovo ispravljanje
- **Neovisni nadzor** - Poštivanje ovih pravila nadzire neovisno tijelo

Kako bi se osigurala provedba ovih prava, Europska unija je usvojila Opću uredbu o zaštiti podataka (GDPR). Ovaj ključni zakonodavni okvir postavlja stroge zahtjeve za organizacije koje prikupljaju i obrađuju osobne podatke. Organizacije su obvezne dobiti jasnu, informiranu i nedvosmislenu privolu od pojedinaca prije prikupljanja podataka, a pojedinci imaju pravo pristupiti svojim podacima, zahtijevati njihovu ispravku ili brisanje kada podaci više nisu potrebni ili kada povuku pristanak. Nadalje, GDPR omogućuje pojedincima da ograniče obradu svojih podataka te zatraže njihovu prenosivost između organizacija u strojno čitljivom formatu. U slučaju povrede podataka, organizacije moraju obavijestiti nadzorno tijelo unutar 72 sata, a u nekim slučajevima i same pojedince, kako bi se minimizirala šteta i osigurala odgovornost za zaštitu osobnih podataka. Kako bi se osigurala usklađenost s ovim pravilima, nadzorna tijela u svakoj zemlji članici EU imaju ovlast provoditi neovisni nadzor, uključujući mogućnost izricanja značajnih kazni za kršenje uredbe.[8] Ovi zahtjevi postavljaju visoke standarde transparentnosti i sigurnosti, te su ključni za zaštitu prava pojedinaca u digitalnom dobu.

2.4 Tehnologije za poboljšanje privatnosti

Prema definiciji Agencije Europske unije za kibernetičku sigurnost (The European Union Agency for Cybersecurity (ENISA)), tehnologije za poboljšanje privatnosti (Privacy Enhancing Technologies (PETs)) predstavljaju softverska i hardverska rješenja, odnosno sustave koji obuhvaćaju tehničke procese, metode ili znanja za postizanje specifične funkcionalnosti privatnosti ili zaštite podataka ili za zaštitu od rizika za privatnost pojedinca ili grupe fizičkih osoba.[4] Drugim riječima, to su ključni alati u modernoj zaštiti privatnosti i podataka, koji omogućuju organizacijama da ispunjavaju sve strože regulative o zaštiti podataka, istovremeno osiguravajući sigurnost

i povjerenje korisnika. Njihovim korištenjem organizacije mogu učinkovito minimizirati rizike povezane s neovlaštenim pristupom ili zloupotrebom podataka, čime se povećava otpornost na kibernetičke prijetnje i smanjuju potencijalni troškovi povezanih incidenata.

2.4.1 Kategorizacija

Kategorizacija je od ključne važnosti za primjenu PETs-a u skladu s međunarodnim standardima i regulativama, te služi kao vodič organizacijama u odabiru optimalnih tehnologija za očuvanje privatnosti. Učinjeno je nekoliko pokušaja klasificiranja i kategoriziranja PETs-a na temelju temeljne tehnologije ili slučajeva upotrebe na koje se odnose. Najnovija kategorizacija PETs-a, razvijena od strane Organizacije za ekonomsku suradnju i razvoj (Organisation for Economic Co-operation and Development (OECD)), detaljno opisuje različite vrste ovih tehnologija, njihove primjene i ograničenja u zaštiti podataka i privatnosti. Ova podjela omogućuje preciznije razumijevanje specifičnih funkcionalnosti svakog rješenja te je prikazana na Slici 2.1.

PETs su podijeljeni u četiri glavne kategorije: alati za obfuscaciju podataka, alati za obradu podataka kroz enkripciju, distribuirana i federirana analitika te alati za odgovornost podataka. Svaka kategorija obuhvaća različite tehnologije, kao što su anonimizacija, homomorfna enkripcija, distribuirano učenje i sustavi za odgovornost podataka. Primjene PETs-a obuhvaćaju širok raspon aktivnosti, uključujući sigurnu pohranu podataka, zaštitu privatnosti u strojnom učenju i verifikaciju informacija bez otkrivanja dodatnih podataka. Unatoč brojnim prednostima, ove tehnologije suočavaju se s raznim izazovima, poput visokih troškova računalne obrade, složenosti konfiguracije i digitalnih sigurnosnih izazova.

Alati za obfuscaciju podataka transformiraju podatke lokalno, na uređaju korisnika, dodajući "šum" ili uklanjajući identifikacijske detalje kako bi se zaštitila privatnost. Alati za obradu podataka kroz enkripciju omogućuju izvođenje proračuna nad podacima koji ostaju skriveni kroz enkripciju, bez izmjene samih podataka. Ova tehnologija predstavlja značajan iskorak u privatnoj obradi podataka, iako je tek nedavno postala izvediva zahvaljujući napretku računalne snage. Distribuirana i federirana analitika omogućuje izvršavanje analitičkih zadataka nad podacima koji

Types of PETs	Key technologies	Current and potential applications*	Challenges and limitations
Data obfuscation tools	Anonymisation / Pseudonymisation	Secure storage	- Ensuring that information does not leak (risk of re-identification) - Amplified bias in particular for synthetic data - Insufficient skills and competences
	Synthetic data	Privacy-preserving machine learning	
	Differential privacy	Expanding research opportunities	
	Zero-knowledge proofs	Verifying information without requiring disclosure (e.g. age verification)	- Applications are still in their early stages
Encrypted data processing tools	Homomorphic encryption	Computing on encrypted data within the same organisation Computing on private data that is too sensitive to disclose Contact tracing / discovery	- Data cleaning challenges - Ensuring that information does not leak - Higher computation costs
	Multi-party computation (including orivate set intersection)		
	Trusted execution environments	Computing using models that need to remain private	- Higher computation costs - Digital security challenges
Federated and distributed analytics	Federated learning	Privacy-preserving machine learning	- Reliable connectivity needed - Information on data models need to be made available to data processor
	Distributed analytics		
Data accountability tools	Accountable systems	Setting and enforcing rules regarding when data can be accessed Immutable tracking of data access by data controllers	- Narrow use cases and lack stand-alone applications - Configuration complexity - Privacy and data protection compliance risks where distributed ledger technologies are used - Digital security challenges - Not considered as PETs in the strict sense
	Threshold secret sharing		
	Personal data stores / Personal Information Management Systems	Providing data subjects control over their own data	

Slika 2.1 Pregled glavnih vrsta PETs-a [5]

ostaju pod nadzorom izvora podataka, dok se samo sažeti statistički podaci ili rezultati prenose trećim stranama koje provode analizu. Alati za odgovornost podataka pružaju nove kontrole nad prikupljanjem i korištenjem podataka, kao i transparentnost u transakcijama. Iako se tradicionalno ne smatraju PETs-ovima u užem smislu, često su povezani s njima jer omogućuju jaču privatnost kroz regulaciju i kontrolu nad obradom podataka.[5]

2.4.2 Primjena

PETs značajno doprinose zaštiti osjetljivih podataka u različitim industrijama. Primjerice, u financijskom sektoru omogućuju sigurnu analizu podataka radi otkrivanja prijevara bez kompromitiranja privatnosti korisnika. U zdravstvu, ove tehnologije omogućuju sigurno dijeljenje i analizu medicinskih podataka, što potiče napredak u istraživanjima i očuvanje privatnosti pacijenata. U maloprodaji, PETs omogućuju analiziranje potrošačkih podataka za marketinške svrhe, omogućujući tvrtkama da personaliziraju ponude bez ugrožavanja privatnosti kupaca. U kibernetičkoj sigurnosti, pomažu u zaštiti osjetljivih sigurnosnih podataka, otkrivaju prijetnje i analiziraju mrežni promet bez izlaganja ranjivosti. Također, u digitalnom marketingu, PETs pomažu u prikupljanju i analiziranju podataka o korisnicima za ciljanje oglasa, dok se istovremeno čuva privatnost.[6] Ovo su samo neke od najkorištenijih primjena ovih tehnologija, dok postoji mnoštvo drugih mogućnosti i područja njihove upotrebe.

Poglavlje 3

Secure Multiparty Computation (MPC)

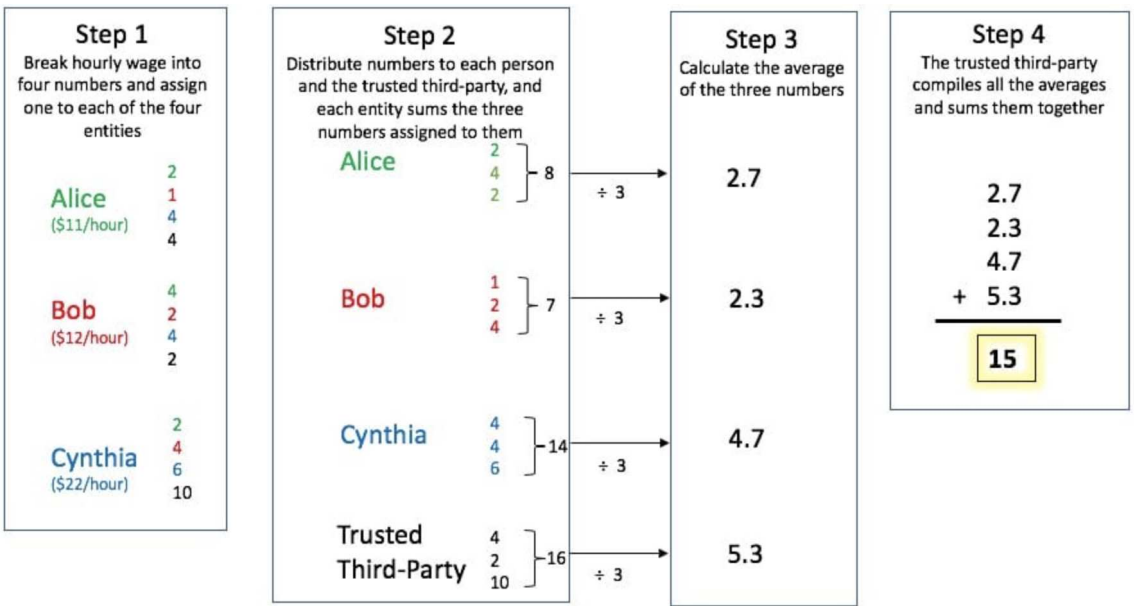
Jedna od ključnih tehnologija za poboljšanje privatnosti u modernim digitalnim sustavima je MPC. MPC koristi kriptografske primitive poput dijeljenja tajni (npr. Shamir), homomorfne enkripcije (npr. Paillier, ElGamal), i metode *zero-knowledge proof* (npr. zk-SNARKs, zk-STARKs) kako bi omogućio određenom broju (n) sudionika, od kojih svaki posjeduje privatne podatke (d1, d2, ..., dn), da izračunaju javnu funkciju na tim podacima F(d1, d2, ..., dn), bez da ijedan sudionik sazna informacije o unosima drugih sudionika.[9] Drugim riječima, MPC omogućava grupi sudionika da zajednički obrade podatke i dođu do određenog rezultata, a da pritom svaki sudionik zadrži povjerljivost svojih podataka. Na taj način informacije se dijele i obrađuju na siguran način, bez izlaganja privatnih podataka riziku.

Kako bi osigurali sigurnu i učinkovitu obradu podataka, MPC protokoli moraju jamčiti nekoliko temeljnih svojstava. Privatnost osigurava da niti jedan sudionik ne može saznati bilo kakve informacije o tuđim podacima tijekom izvršenja protokola, osim onoga što se može zaključiti iz vlastitog unosa i rezultata izračuna. Ispravnost jamči da svaki sudionik dobije točan rezultat, bez mogućnosti da se taj rezultat promijeni ili manipulira tijekom obrade. Neovisnost ulaza znači da kompromitirane strane ne mogu prilagoditi svoje ulaze na temelju ulaza drugih sudionika, čime se sprječava utjecaj na ishod procesa. Zajamčena isporuka rezultata osigurava da nijedan

sudionik ne može onemogućiti drugima pristup krajnjem rezultatu izračuna. Naposlijetku, pravednost osigurava da svi sudionici dobiju svoje rezultate istovremeno, bez mogućnosti da jedna strana dobije svoj rezultat prije drugih.[10]

MPC rješava problem izračuna agregatnih statistika s očuvanjem privatnosti tako što omogućava da se osjetljivi podaci obrađuju bez njihovog otkrivanja bilo kojoj pojedinačnoj strani. U tradicionalnim metodama, sudionici bi morali dijeliti svoje podatke s centraliziranom trećom stranom koja bi zatim izvršila izračune, što nosi rizik od curenja ili zloupotrebe podataka. Nasuprot tome, MPC omogućava da svaki sudionik podijeli svoje podatke u šifriranom ili fragmentiranom obliku s ostalim sudionicima, tako da nitko ne može vidjeti kompletne podatke bilo kojeg drugog sudionika. Ovi fragmenti se zatim koriste za zajedničko izračunavanje željene statistike, kao što su prosjeci ili zbrojevi, pri čemu se svi izračuni odvijaju bez dešifriranja podataka. Na taj način, konačni rezultat je dostupan svim sudionicima, dok su individualni unosi ostaju zaštićeni, čime se efikasno štiti privatnost podataka tijekom cijelog procesa.

Jedan od jednostavnijih primjera izračuna agregatne statistike pomoću MPC-a prikazan je na Slici 3.1. te ilustrira proces izračuna prosječne satnice triju zaposlenika. Proces je podijeljen u četiri koraka. U prvom koraku, satnica svakog zaposlenika (Alice, Bob i Cynthia) se razdvaja na četiri broja koji su zatim raspodijeljeni između sudionika i pouzdane treće strane. U drugom koraku, svaki sudionik i pouzdana treća strana zbrajaju njima dodijeljene brojeve. U trećem koraku, svaki od sudionika izračunava prosjek svog zbroja podijelivši ga s brojem sudionika. U završnom koraku, pouzdana treća strana prikuplja te prosjeke i zbraja ih kako bi dobila ukupni rezultat od 15 dolara. Ovaj proces omogućava izračun prosječne satnice bez otkrivanja pojedinačnih satnica bilo kojeg sudionika, što ilustrira ključnu prednost MPC-a u zaštiti privatnosti podataka.



Slika 3.1 Primjer izračuna agregatne statistike pomoću MPC-a [11]

3.1 Povijest i razvoj

Povijest i razvoj MPC-a započinju u ranim 1980-ima, kada su istraživači počeli istraživati načine na koje više strana može zajedno izračunati neku funkciju na temelju svojih privatnih ulaza, a da pritom ne otkrivaju te ulaze jedni drugima. Prvi korak u razvoju MPC-a napravio je Andrew Yao u svom radu iz 1986. godine "How to Generate and Exchange Secrets", u kojem je uveo koncept sigurnog dvočlanog računanja (engl. *two-party computation*). Yao je osmislio protokol koji omogućuje dvjema stranama da izračunaju rezultat funkcije bez otkrivanja svojih privatnih podataka, što je postalo poznato kao Yaoov protokol.

Ovaj protokol radi ovako: Prvo, jedna strana (pošiljatelj) stvara šifriranu verziju funkcije koju žele izračunati, koja je predstavljena kao niz enkriptiranih koraka za računanje. Pošiljatelj zatim šalje ovu šifriranu verziju drugoj strani (primatelju) zajedno s informacijama potrebnim za izračun. Druga strana, koja posjeduje svoje privatne podatke, koristi specijalnu metodu zvanu oblivious transfer kako bi odabrala potrebne dijelove šifrirane verzije funkcije bez da pošiljatelj sazna koje dijelove je

odabrala. Na taj način, primatelj može obaviti izračun i saznati ishod (primjerice tko ima više novaca) bez da pošiljatelj sazna bilo kakve informacije o privatnim podacima primatelja.[12]

Nakon što je Yao postavio temelje za sigurno dvočlano računanje, istraživači su dalje razvijali ovu ideju za složenije scenarije sa više sudionika, pri čemu je važan doprinos dao rad iz 1987. godine: "How to Play ANY Mental Game or A Completeness Theorem for Protocols with Honest Majority". Autori su proširili Yaoove ideje i pokazali da ako većina sudionika u grupi djeluje ispravno, moguće je izračunati bilo koju funkciju koristeći tajne podatke svih sudionika, bez otkrivanja privatnih informacija. Ovaj rad uvodi ključne koncepte poput tajnog dijeljenja podataka i metode *zero-knowledge proof* kako bi se osigurala privatnost i točnost u složenim scenarijima. Tajno dijeljenje podataka omogućava da se ulazne informacije podijele na način koji skriva podatke od svih osim od ovlaštenih sudionika, dok *zero-knowledge proof* omogućuje sudionicima da potvrde ispravnost tvrdnji bez otkrivanja dodatnih informacija. Ove metode su osobito korisne u situacijama gdje može doći do zlonamjernih ponašanja, jer omogućuju većini poštenih sudionika da otkriju takvo ponašanje i nastave s računom, eliminirajući nepoštene sudionike ili otkrivajući njihove podatke.[13]

U godinama nakon, istraživanja su se usmjerila na prilagodbu MPC protokola za širi raspon aplikacija i situacija koje uključuju moguće zlonamjerne sudionike. Od početka 2000-ih, fokus je bio na razvoju učinkovitijih protokola koji se mogu primijeniti u stvarnim uvjetima, čime je MPC postao rješenje za različite probleme u stvarnom svijetu. U 2020. godini, osnovan je MPC savez s ciljem da ubrza usvajanje i primjenu MPC tehnologije, čime se ističe rastući značaj i utjecaj ove tehnologije u industriji i svakodnevnim praksama.[14]

3.2 Prednosti

MPC donosi značajne prednosti u obradi podataka, prvenstveno kroz poboljšanu sigurnost i otpornost na napade. Tradicionalni pristupi obradi podataka često uključuju prenošenje i pohranu osjetljivih informacija na središnjim serverima, što može

predstavljati sigurnosni rizik. U MPC-u, podaci se dijele na tajne dijelove i obrađuju u šifriranom obliku među različitim sudionicima, čime se drastično smanjuje rizik od neovlaštenog pristupa i curenja informacija. Ova metoda omogućava da podaci ostanu povjerljivi čak i u slučaju da neki sudionici u sustavu nisu u potpunosti pouzdani, što značajno poboljšava ukupnu sigurnost obrade podataka. Uz to, MPC je otporan na kvantne napade jer su podaci tijekom obrade raspodijeljeni, što ih čini sigurnima čak i u eri kvantnih računala, osiguravajući dugoročnu zaštitu informacija.

Osim što poboljšava sigurnost, MPC omogućava očuvanje privatnosti korisnika bez kompromisa u funkcionalnosti i točnosti rezultata. U mnogim tradicionalnim sustavima, kako bi se zaštitila privatnost, često je potrebno maskirati ili ukloniti određene podatke, što može smanjiti točnost analiza. MPC omogućava korištenje svih značajki podataka u analizi, dok istovremeno osigurava da privatni podaci ostanu zaštićeni. Rezultati dobiveni pomoću MPC ne samo da zadovoljavaju, već često i premašuju zahtjeve za točnošću i preciznošću, što je ključno u industrijama kao što su financije i zdravstvo, gdje je visoka razina povjerljivosti i detaljna analiza nužna za donošenje informiranih odluka.

MPC također igra ključnu ulogu u ispunjavanju regulatornih zahtjeva kao što je GDPR. Procesiranje osjetljivih podataka bez njihovog izlaganja vanjskim entitetima omogućava organizacijama da se usklade s strogim propisima o zaštiti podataka. Budući da podaci nikada ne napuštaju kontrolirane okvire organizacije i obrađuju se u šifriranom obliku, SMPC smanjuje rizik od neusklađenosti i kazni, čime se omogućava lakše pridržavanje pravila i zakona koji reguliraju zaštitu privatnosti.

3.3 Izazovi i ograničenja

Unatoč značajnim prednostima, MPC se suočava s brojnim izazovima i ograničenjima koji otežavaju njegovu široku primjenu. Jedan od glavnih izazova su zlonamjerni sudionici unutar sustava. Većina MPC protokola temelji se na pretpostavci da će sudionici poštovati pravila i pošteno slijediti protokol. No, ako čak i jedan sudionik postane zlonamjeran i odstupi od protokola, može ugroziti sigurnost cijelog sustava. Zlonamjerni sudionici mogu pokušati prikupiti više informacija nego što bi trebali,

ili manipulirati ishodom računanja u svoju korist. Ovi scenariji zahtijevaju dodatne slojeve sigurnosti kako bi se spriječilo takvo zlonamjerno ponašanje, što često rezultira dodatnim računalnim i komunikacijskim troškovima.

Osim prijetnji unutar sustava, MPC protokoli su podložni napadima putem bočnih kanala, kao što su mjerenje potrošnje energije, praćenje vremena izvršavanja ili elektromagnetska zračenja. Takvi napadi mogu neizravno otkriti osjetljive informacije o izračunima, kompromitirajući sigurnost podataka. Iako su MPC protokoli dizajnirani da štite privatnost tijekom računanja, bočni kanali predstavljaju rizik koji je teško potpuno eliminirati. Uz bočne kanale, tu su i skriveni kanali putem kojih zlonamjerni sudionici mogu komunicirati neprimjetno i zaobići sigurnosne mjere sustava.

Budući da MPC zahtijeva značajne računalne resurse, to ga čini ranjivim na napade usmjerene na iscrpljivanje resursa. Napadači mogu pokušati preopteretiti sustav pretjeranim zahtjevima za računalne ili komunikacijske resurse, uzrokujući degradaciju performansi ili čak potpunu nemogućnost obavljanja računanja. Kako bi se spriječile ove ranjivosti, ključno je pažljivo dizajnirati i implementirati MPC protokole, provoditi detaljne sigurnosne provjere te kontinuirano nadzirati sustav za potencijalne napade.

3.4 Primjena

Prva praktična primjena MPC-a ostvarena je u siječnju 2008. godine kroz implementaciju sigurnih aukcija u Danskoj. Ova tehnologija korištena je za provedbu dvostruke aukcije u sektoru proizvodnje šećerne repe, gdje su poljoprivrednici mogli prodavati ili kupovati prava na proizvodnju. Ponude poljoprivrednika bile su privatne, a MPC protokol osigurao je da se proces aukcije odvija na način koji čuva povjerljivost ponuda, bez potrebe za povjerenjem u treću stranu kao što je Danisco, jedina tvrtka koja prerađuje šećernu repu u zemlji. Protokol je također bio odgovoran za određivanje najveće ponude, pri čemu je poljoprivrednik s pobjedničkom ponudom platio iznos druge najveće ponude u aukciji. Ovaj sustav omogućio je sigurnu i transparentnu redistribuciju proizvodnih prava, a istovremeno je riješio potenci-

jalne sigurnosne probleme koji bi mogli utjecati na način na koji poljoprivrednici sudjeluju u aukciji. Usprkos tehničkim izazovima, MPC se pokazao kao djelotvorno rješenje koje je zadovoljilo sve sudionike u procesu, a ova prva velika implementacija MPC-a otvorila je vrata za daljnje praktične primjene ove tehnologije u različitim sektorima.[15]

Osim primjene u aukcijama, jedna od ključnih primjena MPC-a danas je analiza podataka uz očuvanje privatnosti. U industrijama kao što su zdravstvena zaštita, financije i marketing, gdje je analiza osjetljivih podataka često neophodna, MPC može omogućiti suradnju između različitih organizacija bez otkrivanja pojedinačnih podataka. MPC se također može koristiti u strojnom učenju za obuku modela s osjetljivim podacima iz više izvora, bez otkrivanja tih podataka. Primjerice, tvrtke mogu koristiti MPC kako bi kombinirale podatke iz financijskih institucija i maloprodajnih lanaca kako bi razvile bolje algoritme za prepoznavanje obrazaca, dok se privatnost korisnika čuva. U području otkrivanja i prevencije prijevara, MPC može omogućiti sigurno analiziranje transakcijskih podataka iz više izvora. U bankarskom sektoru, MPC može pomoći u otkrivanju sumnjivih aktivnosti analizom podataka iz različitih banaka bez otkrivanja osobnih podataka korisnika, čime se poboljšava točnost sustava za otkrivanje prijevara. Nadalje, računarstvo u oblaku može koristiti MPC za izvođenje izračuna na šifriranim podacima, što omogućava organizacijama da koriste prednosti oblaka dok zadržavaju kontrolu nad podacima. Također, MPC se može koristiti kod upravljanja identitetom uz očuvanje privatnosti za razvoj sigurnih sustava autentifikacije, uključujući e-glasanje i online autentifikaciju. Ova primjena omogućava verifikaciju identiteta korisnika pomoću biometrijskih podataka bez otkrivanja tih podataka, poboljšavajući sigurnost i povjerenje u procese poput glasanja.

Poglavlje 4

Arhitektura aplikacije

Ova aplikacija zamišljena je kao istraživanje koje provodi Tehnički fakultet u Rijeci kao davatelj usluga. Cilj istraživanja je otkriti najčešće i najrjeđe napade te utvrditi postoji li povezanost frekvencije pojavljivanja kibernetičkih napada i vremenskog perioda u kojem se pojavljuju. U tu svrhu, od tvrtki koje sudjeluju u istraživanju se traže podaci o učestalosti 10 različitih kibernetičkih napada tijekom razdoblja od 12 mjeseci. Kako bi se osigurala vjerodostojnost podataka, prilikom registracije svaka tvrtka upisuje svoj naziv i email. Ova informacija pomaže u verifikaciji identiteta i osigurava da samo legitimne tvrtke sudjeluju u istraživanju. Osim toga, ova mjera pomaže u prevenciji zloupotreba, poput lažnih prijava ili aktivnosti botova koji bi mogli neautorizirano objavljivati podatke. Podaci o nazivima i emailovima tvrtki koje sudjeluju u istraživanju nikad se ne dijele sa analitičarom.

Rješavanje problema izračuna agregatnih statistika uz očuvanje privatnosti podataka u ovoj aplikaciji ostvaruje se korištenjem MPC protokola. Ključna snaga ovog pristupa leži u njegovoj sposobnosti da omogući izračunavanje agregatnih statistika bez potrebe za otkrivanjem pojedinačnih podataka bilo kojoj trećoj strani tijekom cijelog procesa. Budući da ni davatelj usluga ni analitičar ne može vidjeti stvarne podatke, rizik od otkrivanja ili zloupotrebe osjetljivih informacija je minimiziran. Također, ovakav pristup zadovoljava zakonske okvire i propise o zaštiti podataka jer osigurava da se podaci prikupljaju i obrađuju na način koji jamči privatnost i povjerljivost.

Polazišna točka arhitekture ove aplikacije bio je znanstveni rad "From usability to secure computing and back again" objavljen 2019. godine na Petnaestom simpoziju o upotrebljivoj privatnosti i sigurnosti (SOUPS 2019). U radu je opisana aplikacija koja istražuje i demonstrira kako se principi sigurnosti mogu implementirati u svakodnevne računalne sisteme bez narušavanja njihove upotrebljivosti. Aplikacija je primijenjena u dva specifična scenarija: evaluacija plaća 166 705 zaposlenika u Bostonu i mjerenje stope po kojoj članice Gospodarske komore Velikog Bostona sklapaju podugovore s poduzećima u vlasništvu manjina. U aplikaciji, analitika sa strane klijenta se prikuplja putem postojećeg MPC protokola. Podaci su maskirani i poslani davatelju usluga, dok su maske enkriptirane i poslane analitičaru, čime se osigurava da nijedna informacija ne može biti vezana uz specifičnog korisnika.

Rad pokazuje da se MPC može uspješno koristiti za prikupljanje analitičkih podataka uz očuvanje privatnosti, ali također ističe potrebu za daljnjim unapređenjima u dizajnu i implementaciji kako bi se smanjili troškovi i olakšala upotreba aplikacije. Također naglašava važnost obuke korisnika i jasnih prikaza funkcionalnosti protokola kako bi se osiguralo povjerenje i pravilna upotreba u stvarnim situacijama.[16]

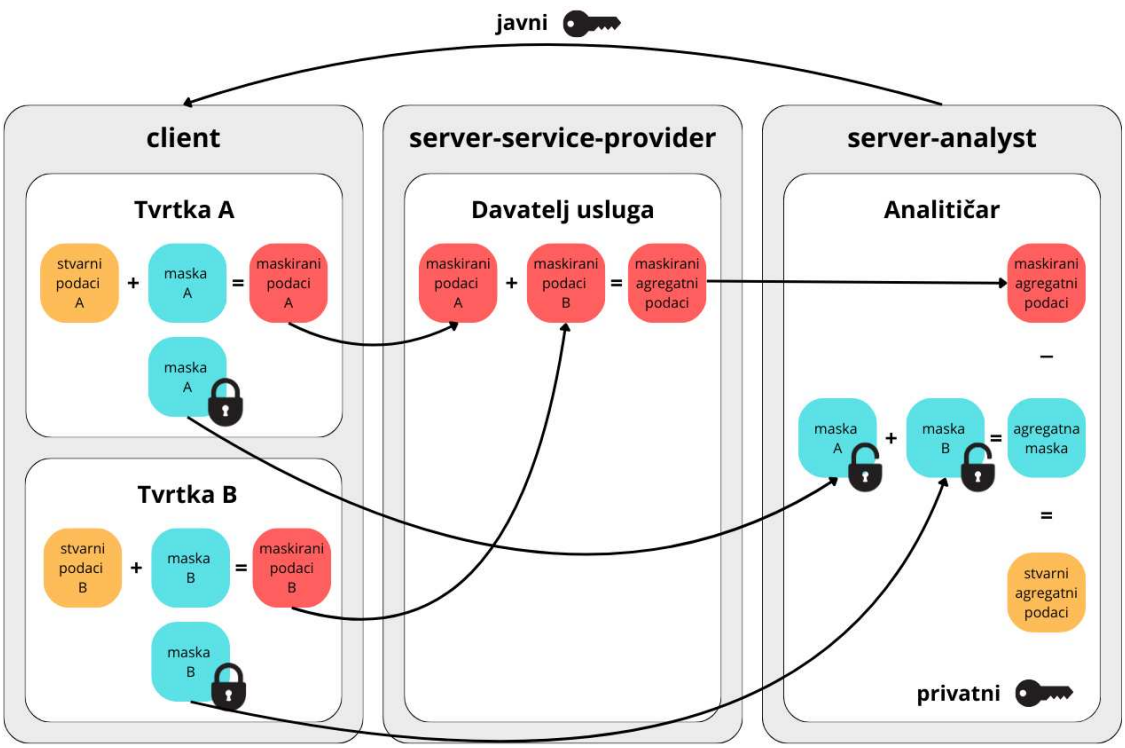
4.1 Pregled sustava

Funkcijski dijagram aplikacije ovog diplomskog rada prikazan je na Slici 4.1. Na početku, poslužitelj analitičar šalje svoj javni ključ klijentskoj aplikaciji. Klijentska aplikacija se nalazi u lijevom dijelu dijagrama i predstavljena je kao Tvrtka A i Tvrtka B. Svaka tvrtka unosi stvarne podatke o kibernetičkim napadima u sustav, koji se zatim maskiraju koristeći jedinstvenu masku za svaki unos. Maska štiti stvarne podatke, tako da čak i ako maskirani podaci budu presretnuti, ne mogu se povezati sa stvarnim informacijama. Maskirani podaci šalju se poslužitelju davatelju usluga, dok se korištena maska enkriptira pomoću javnog ključa i šalje poslužitelju analitičaru.

Nakon što se maskirani podaci pošalju davatelju usluga, on prikuplja ove maskirane podatke od svih tvrtki koje sudjeluju u analizi. Budući da su podaci maskirani, davatelj usluga ne može vidjeti stvarne podatke koje su tvrtke unijele, čime se osigurava privatnost svakog pojedinačnog unosa. Davatelj usluga zatim zbraja sve podatke

po odgovarajućim poljima, budući da su maskirani podaci u formi dvodimenzionalne numeričke matrice. Dobiveni maskirani agregatni podaci se zatim šalju analitičaru.

Analitičar prima ove maskirane agregatne podatke zajedno s enkriptiranim maskama koje su korištene za enkripciju podataka od strane svake tvrtke. Prvi korak koji analitičar poduzima je dekripcija maski pomoću svog privatnog ključa. Nakon što dekriptira maske, analitičar ih zbraja kako bi dobio tzv. agregatnu masku. Naposlijetku, analitičar oduzima agregatnu masku od maskiranih agregatnih podataka kako bi dobio stvarne agregatne podatke. Važno je napomenuti da iako analitičar sada može vidjeti stvarne agregatne podatke, individualni podaci koje su unijele pojedinačne tvrtke i dalje ostaju privatni. S obzirom na to da su podaci kombinirani tek u agregatnom obliku, informacije o specifičnim unosima pojedinih tvrtki nisu izravno dostupne niti se mogu rekonstruirati iz agregatnih podataka.



Slika 4.1 Funkcijski dijagram.

4.2 Baza podataka

Aplikacija komunicira sa dvije baze podataka: *serviceProvider* i *analyst*. Davatelj usluga ima pristup isključivo bazi podataka *serviceProvider* i nema mogućnost pristupa ili uvida u podatke koji se nalaze u bazi podataka *analyst*. Na isti način, analitičar je povezan samo na bazu *analyst* i može raditi isključivo s podacima unutar te baze, bez mogućnosti interakcije s podacima iz baze *serviceProvider*. Ovaj pristup osigurava jasnu granicu između različitih poslužitelja i njihovih odgovornosti, čime se povećava sigurnost i kontrola pristupa podacima. Budući da je korišten MongoDB kao NoSQL baza podataka, baze su organizirane kao skup kolekcija, dok kolekcije pohranjuje podatke u obliku dokumenata. Svaki dokument unutar kolekcije je strukturiran kao JSON objekt i sadrži parove ključ-vrijednost. Detaljnija objašnjenja o prednostima MongoDB-a i razlozima za njegov odabir bit će obrađena u sljedećem poglavlju.

Prateći ovu strukturu, u bazi *serviceProvider* se nalaze dvije kolekcije: *users* i *masked_data*. Kolekcija *users* sadrži podatke o nazivu, emailu i hashiranoj lozinci, dok kolekcija *masked_data* sadrži ID korisnika koji je poslao podatke te podatke maskirane jedinstvenom maskom i organizirane u dvodimenzionalnu numeričku matricu. U bazi *analyst* se također nalaze dvije kolekcije: *encrypted_masks* i *encrypted_aggregate_data*. Kolekcija *encrypted_masks* sadrži ID korisnika i enkriptiranu masku koja je korištena za maskiranje podataka korisnika., a kolekcija *encrypted_aggregate_data* sadrži samo enkriptirane finalne agregatne podatke.

Bitno je napomenuti da kada korisnik unese nove podatke u sustav, sustav provjerava je li taj korisnik već ranije unosio podatke. Ukoliko je već unio podatke, umjesto stvaranja novog zapisa u bazi podataka, postojeći zapis se ažurira s novim podacima. Ova strategija osigurava da se pohranjuje samo jedan zapis po korisniku, koji uvijek odražava najnovije podatke koje je korisnik unio. Korisnički ID (*userId*) pohranjuje se uz podatke kako bi se osiguralo da se za finalne agregatne podatke uzima u obzir samo posljednji unos svakog korisnika. Na taj način sprječava se dvostruko ili višestruko ubrajanje podataka ako bi korisnik nekoliko puta unio ili ažurirao podatke.

4.3 Autentifikacija i autorizacija

Autentifikacija i autorizacija su ključni elementi sigurnosti u svakoj web aplikaciji. Autentifikacija potvrđuje identitet korisnika, osiguravajući da samo ovlašteni korisnici mogu pristupiti sustavu, dok autorizacija određuje koje resurse i funkcionalnosti korisnik može koristiti nakon što je autentificiran. U ovoj aplikaciji, ovi procesi su implementirani kroz kombinaciju frontend i backend validacija, sigurnog hashiranja lozinki, generiranja i upravljanja JWT tokenima te korištenja next-auth knjižnice za upravljanje sesijama.

Validacija korisničkih podataka odvija se na obje strane aplikacije kako bi se osigurala cjelovitost i sigurnost informacija koje korisnici unose. Frontend validacija koristi Zod knjižnicu za definiranje shema i pravila validacije unutar formi za registraciju i prijavu. Primjerice, za registracijsku formu definirana je shema koja provjerava je li email u ispravnom formatu, lozinka ima minimalno 8 znakova, a ime korisnika sadrži između 2 i 30 slova. Ova validacija pruža korisniku brze povratne informacije i sprječava nepotrebne zahtjeve prema serveru s neispravnim podacima. Backend validacija dodatno provjerava podatke te osigurava da ne postoji već registrirani korisnik s istim emailom u bazi podataka. Ova dvostruka validacija sprječava potencijalne zlonamjerne pokušaje zaobilaženja sigurnosnih mjera putem manipulacije frontend koda.

Nakon što korisnički podaci prođu validaciju, prilikom registracije, lozinke se sigurno pohranjuju u bazu podataka koristeći bcrypt. Bcrypt je robustan algoritam za hashiranje koji dodaje nasumični "salt" svakoj lozinci i provodi više rundi hashiranja (u ovom slučaju 10 rundi), što značajno otežava pokušaje brute-force napada i obrnutog inženjeringa lozinki iz hashiranih vrijednosti. Time se osigurava da čak i u slučaju kompromitacije baze podataka, originalne lozinke korisnika ostanu zaštićene.

Prilikom prijave u aplikaciju, korisnik unosi svoj email i lozinku, koje se šalju poslužitelju davatelju usluga radi provjere autentičnosti. Poslužitelj traži korisnika u bazi podataka na temelju unesenog emaila. Ako korisnik postoji, unijeta lozinka se uspoređuje s hashiranom lozinkom pohranjenom u bazi. Kada je provjera uspješna, generira se JSON Web Token (JWT), koji sadrži ID, ime i email korisnika. JWT je digitalno potpisan token koji se koristi za za sigurno prenošenje informacija između

klijenta i poslužitelja. Sastoji se od tri dijela: zaglavlja, podataka (payload) i potpisa, koji su enkodirani u Base64 formatu. Token se prilikom prijave šalje klijentu koji ga pohranjuje i koristi za autentifikaciju budućih zahtjeva prema serveru, čime se izbjegava potreba za ponovnim unosom podataka pri svakom zahtjevu. Valjanost tokena je ograničena (u ovom slučaju jedan dan), što dodatno povećava sigurnost sustava.

Nakon uspješne prijave, klijent dobiva JWT od poslužitelja i dekodira ga kako bi dobio korisničke podatke. Korisnički podaci i JWT se pohranjuju u objekt, koji se zatim prenosi u sesiju. Sesija predstavlja mjesto gdje se pohranjuju svi relevantni podaci potrebni za upravljanje korisničkim iskustvom tijekom trajanja prijave. U ovoj fazi, sesija uključuje sve informacije iz JWT-a (korisnički ID, ime i email), zajedno s JWT-jem koji omogućava pristup zaštićenim resursima. Ove informacije su ključne za personalizaciju korisničkog iskustva i za upravljanje pravima pristupa unutar aplikacije. Također, ovaj proces osigurava da su svi relevantni podaci o korisniku dostupni i ažurirani tijekom trajanja sesije.

Autorizacija je implementirana putem middlewarea koji se koristi za kontrolu pristupa na temelju JWT-a. Kada dolazi zahtjev, middleware prvo provjerava putanju zahtjeva. Ako je putanja među onima koje su dopuštene bez autentifikacije ("/login", "/register"), zahtjev se odmah prosljeđuje za daljnju obradu. Ako putanja zahtjeva nije na popisu, middleware dohvaća JWT korisnika kako bi provjerio njegov status. Ako token ne postoji ili je označen kao neautentificiran, korisnik se preusmjerava na stranicu za prijavu. Ako je korisnik autentificiran, middleware dodatno provjerava je li korisnik analitičar i je li putanja zahtjeva među onima koje su ograničene za analitičare ("/", "/data-entry"). Ako je to slučaj, korisnik se preusmjerava na zadanu dopuštenu stranicu. U svim drugim slučajevima, zahtjev se omogućava i prosljeđuje dalje. Ova logika omogućava preciznu kontrolu pristupa temeljem statusa autentifikacije i specifičnih korisničkih prava.

Poglavlje 5

Programska podrška

5.1 Frontend

Za izradu ove aplikacije odlučila sam koristiti Next.js (verzija 14.2.5) i TypeScript zbog njihovih brojnih prednosti koje su ključne za moderne web aplikacije. Next.js je moćan React framework koji nudi izvanredne performanse zahvaljujući značajkama poput Server-Side Rendering (SSR) i Static Site Generation (SSG), koje znatno doprinose bržem učitavanju stranica i boljoj optimizaciji za tražilice. TypeScript, s druge strane, donosi statičko tipiziranje u JavaScript, što pomaže u otkrivanju grešaka već tijekom razvoja i olakšava održavanje koda.

Odabir Next.js-a i TypeScripta nije samo rezultat njihovih tehničkih prednosti, već i moje želje da stvorim aplikaciju koja je maksimalno optimizirana, brza i prilagođena korisnicima. U konačnici, cilj je bio osigurati da aplikacija bude dugoročno održiva, laka za nadogradnju te da korisnicima pruža najbolje moguće iskustvo.

5.1.1 Next.js

Next.js je open-source framework za React, razvijen od strane Vercel-a, koji omogućuje izradu modernih web aplikacija s naglaskom na performanse, SEO optimizaciju i fleksibilnost. React, kao osnova Next.js-a, ključan je za izradu komponentno baziranih korisničkih sučelja, gdje se aplikacija razbija na manje, izolirane komponente

koje se mogu ponovno koristiti i kombinirati na različitim mjestima unutar aplikacije. Ova modularnost pojednostavljuje razvoj i održavanje koda, jer svaka komponenta može neovisno funkcionirati i biti testirana.

React koristi koncept virtualnog Document Object Model (DOM)-a kako bi optimizirao manipulaciju korisničkim sučeljem. Umjesto da izravno interagira s stvarnim DOM-om, što može biti spor i resursno zahtjevan proces, React stvara virtualnu kopiju DOM-a u memoriji. Ova virtualna verzija DOM-a služi kao privremena reprezentacija stvarnog DOM-a i omogućava Reactu da usporedi trenutne promjene s prethodnim stanjem. Kada dođe do promjene u aplikaciji, React najprije ažurira virtualni DOM i uspoređuje ga s njegovom prethodnom verzijom kako bi identificirao razlike ili promjene. Nakon što se te promjene identificiraju, React primjenjuje samo nužne izmjene na stvarni DOM, umjesto da ponovno renderira cijelo sučelje. Ovaj pristup značajno poboljšava performanse aplikacije, posebno kod velikih i dinamičnih web stranica, jer minimizira broj operacija potrebnih za ažuriranje sučelja.

Next.js dodatno proširuje ove mogućnosti pružajući napredne funkcionalnosti kao što su SSR, Client-Side Rendering (CSR), SSG i Incremental Static Regeneration (ISR). CSR je strategija renderiranja u kojoj se pri prvom zahtjevu učitava minimalan HTML file, dok JavaScript na klijentskoj strani preuzima podatke i generira sadržaj stranice. Drugim riječima, logika aplikacije se učitava i izvršava na strani klijenta, što pruža dinamičniju interaktivnost, ali može rezultirati sporijim početnim učitavanjem. SSR, nasuprot tome, generira HTML sadržaj na serveru za svaki korisnički zahtjev, omogućujući brže prikazivanje stranica i poboljšanu optimizaciju za tražilice. SSG ide korak dalje, generirajući statične HTML stranice unaprijed tijekom build procesa, što omogućuje izuzetno brzo učitavanje i visoku razinu sigurnosti. ISR omogućava regeneraciju dinamičnih dijelova stranice na zahtjev, što znači da određeni dijelovi stranice mogu ostati statični, dok se drugi ponovno renderiraju kad se podaci promijene.[18]

Pored toga, omogućena je jednostavna SEO optimizacija jer se sadržaj stranica prikazuje tražilicama čak i prije nego što se JavaScript učita, što pomaže u boljem rangiranju stranica. Dodatno, Next.js pruža fleksibilnost kroz automatizirano generiranje ruta na temelju strukture datoteka, integraciju s API-ima te podršku za TypeScript, što olakšava razvoj i smanjuje složenost projekata. Zahvaljujući ugra-

đenoj podršci za hot reloading, razvojni proces je značajno ubrzan, jer se promjene odmah reflektiraju u pregledniku bez potrebe za ručnim osvježavanjem stranice.[17]

5.1.2 Typescript

Razvijen od strane Microsoft-a, TypeScript je novi programski jezik koji se nadovezuje na JavaScript pružajući dodatne značajke koje omogućuju bolje upravljanje složenim kodom. Ključna prednost TypeScript-a je njegova sposobnost deklariranja tipova podataka, što omogućuje otkrivanje grešaka tijekom razvoja, prije nego što se aplikacija pokrene. Ovaj sustav tipova omogućuje programerima da eksplicitno definiraju vrste varijabli, funkcija i objekata, čime se značajno smanjuje rizik od runtime grešaka koje se obično javljaju kada aplikacija već radi. Time se poboljšava stabilnost i pouzdanost koda, jer potencijalni problemi mogu biti uočeni i ispravljeni u ranijim fazama razvoja.

Osim toga, TypeScript nudi napredne značajke koje nisu dostupne u standardnom JavaScript-u, poput sučelja i generičkih tipova. Sučelja omogućuju precizno definiranje načina na koji objekti i klase trebaju izgledati, što pomaže u održavanju dosljednosti i jasnoće u kodu. Generički tipovi omogućuju pisanje fleksibilnog i ponovno upotrebljivog koda definiranjem funkcija i klasa koje mogu raditi s različitim vrstama podataka, a da pritom ostanu tipizirane i sigurnosno provjerene.[19] Ovo su samo neke od značajki čine TypeScript izuzetno korisnim za rad na velikim projektima gdje je očuvanje koda u pravilnom i funkcionalnom stanju ključno za dugoročni uspjeh.

5.2 Backend

Za backend razvoj izabrala sam Node.js i Express.js zbog njihove jednostavnosti i sposobnosti da efikasno upravljaju velikim brojem simultanih korisničkih zahtjeva i podataka. Node.js omogućava izgradnju servera i backend sustava koji su brzi, skalabilni i lako integriraju različite web usluge i baze podataka. S druge strane, Express.js, kao lagani i fleksibilni okvir za Node.js, pojednostavljuje izgradnju API-ja i rukovanje HTTP zahtjevima, omogućujući brzo postavljanje ruta i middleware-

a. Budući da su Node.js i Express.js često korišteni zajedno u industriji, njihova integracija pruža čvrstu i pouzdanu osnovu za izgradnju skalabilnih i brzih aplikacija.

5.2.1 Node.js

Node.js je open-source i cross-platformsko JavaScript okruženje koje omogućuje izvršavanje JavaScript koda na serveru, umjesto u web pregledniku. Ovo okruženje, razvijeno na V8 JavaScript engineu, istom onom koji koristi Google Chrome, omogućuje razvoj skalabilnih aplikacija s minimalnim zahtjevima za resurse.[20] Pružajući podršku za moderne web tehnologije i standarde, Node.js omogućuje programerima da koriste jedan jezik za razvoj klijentske i serverske strane aplikacije, što pojednostavljuje razvojni proces i povećava produktivnost.

Jedna od glavnih značajki Node.js-a je njegov pristup upravljanju ulazno-izlaznim operacijama. Node.js ne zaustavlja rad aplikacije dok čeka odgovore s mreže, baze podataka ili datotečnog sustava; umjesto toga, nastavlja s obradom drugih zadataka i vraća se na čekanu operaciju kada odgovor stigne. Ova karakteristika, koja se temelji na asinkronom ulazu/izlazu, omogućuje izuzetnu učinkovitost i skalabilnost aplikacija jer Node.js može upravljati velikim brojem simultanih veza unutar jednog servera bez potrebe za stvaranjem novih dretvi za svaki zahtjev. Ovaj pristup značajno smanjuje potrošnju resursa, što može rezultirati bržim vremenom odziva i smanjenim troškovima operacija.[20]

Uz to, Node.js dolazi s bogatom zbirkom ugrađenih modula koji omogućuju jednostavno rukovanje osnovnim funkcionalnostima poput rada s datotekama, upravljanja mrežnim vezama, kreiranja HTTP servera, i rada s bazama podataka. Paketni menadžer Node Package Manager (npm) dodatno proširuje mogućnosti Node.js-a omogućujući programerima pristup ogromnoj zbirci open-source biblioteka i modula, koje se mogu lako integrirati u projekte. Zahvaljujući ovom bogatom ekosustavu, Node.js je postao jedan od najpopularnijih alata za razvoj backend aplikacija, omogućujući programerima da brzo i efikasno razvijaju složene server-side aplikacije.

5.2.2 Express.js

Express.js je fleksibilan i minimalan web okvir za Node.js koji olakšava izgradnju web aplikacija i API-ja. Kao jedan od najpopularnijih Node.js okvira, Express.js pruža jednostavan način za rukovanje HTTP zahtjevima i odgovorima, upravljanje rutama te integraciju s različitim middleware funkcijama. Kroz svoje bogato API sučelje, Express.js omogućava programerima da kreiraju kompleksne web aplikacije s minimalnim naporom, pružajući alate za definiranje ruta, upravljanje sesijama, autentifikaciju korisnika te rad s raznim vrstama podataka.[21] Bez Express.js okvira, izgradnja backenda od nule u Node.js-u može biti zamorna i vremenski zahtjevna, uključujući postavljanje portova, rukovanje rutama i pisanje boilerplate koda, što oduzima vrijeme koje bi inače bilo usmjereno na ključne funkcionalnosti aplikacije.

Jedna od glavnih prednosti ovog okvira je njegova sposobnost korištenja različitih middleware funkcija. Middleware u Express.js-u su funkcije koje se izvršavaju između primitka HTTP zahtjeva i slanja odgovora te mogu obavljati zadatke poput obrade podataka, provjere autentičnosti korisnika, logiranja zahtjeva, upravljanja sesijama ili obrade grešaka. Ove funkcije omogućuju programerima da prilagode ponašanje aplikacije u različitim fazama obrade zahtjeva. Osim toga, jednostavno je definirati rute za razne HTTP metode (GET, POST, PUT, DELETE).[21]

Primjer korištenja Express.js-a unutar aplikacije je sljedeći isječak programskog koda. Ovaj kod se nalazi u login kontroleru i koristi se za slanje odgovora klijentu nakon uspješne prijave korisnika. Postavljanjem HTTP status koda na 200, što označava uspješan zahtjev, aplikacija zatim šalje JSON odgovor klijentu. Ovaj JSON sadrži dvije ključne informacije: poruku koja informira korisnika da je prijava uspješno obavljena ("Login successful") i JWT koji je prethodno generiran.

```
res.status(200).json({
          message: "Login successful",
          token: jwtToken,
        });
```

5.3 Baza podataka

Za bazu podataka odlučila sam koristiti MongoDB zbog njegove fleksibilnosti i jednostavnosti u radu s različitim vrstama podataka. Za razliku od tradicionalnih relacijskih baza podataka, MongoDB omogućava pohranu podataka u dokumentima koji se mogu jednostavno prilagoditi promjenama. Ova fleksibilnost znatno pojednostavljuje razvoj aplikacije jer više nema potrebe za složenim migracijama baze podataka svaki put kad se struktura podataka mijenja. Uz to, korištenje lokalne instalacije MongoDB-a omogućuje potpunu kontrolu nad konfiguracijom i sigurnošću podataka, što je ključno za očuvanje privatnosti korisnika.

5.3.1 MongoDB

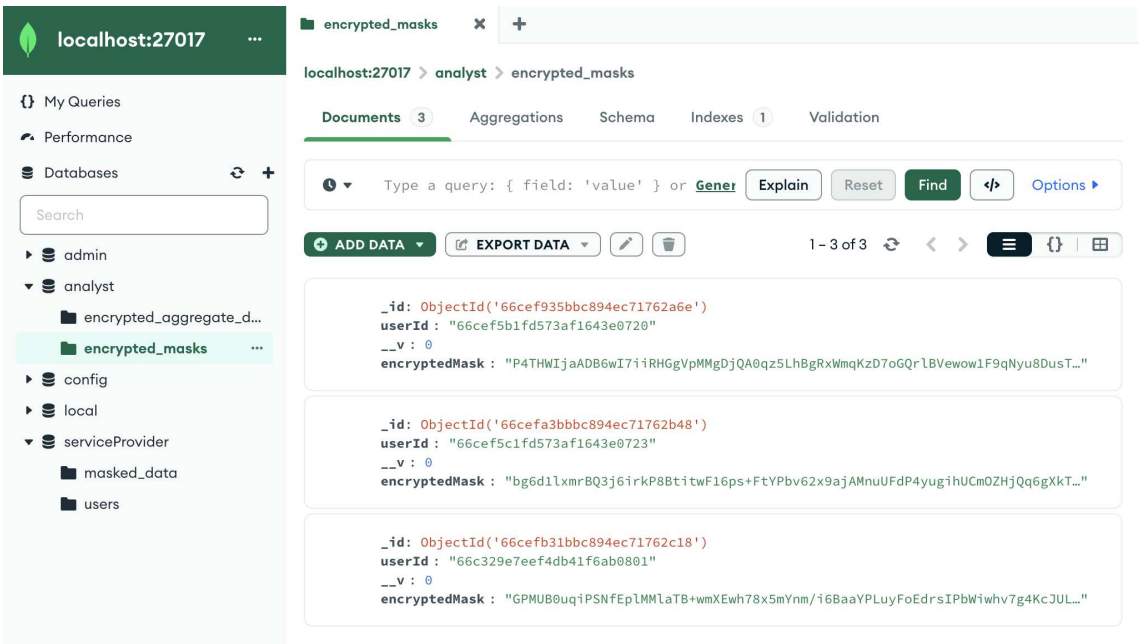
MongoDB je open-source NoSQL baza podataka koja se ističe kao alternativna rješenja u odnosu na tradicionalne relacijske baze podataka. U MongoDB-u, osnovna jedinica podataka je dokument, koji je strukturiran kao parovi ključeva i vrijednosti. Dokumenti su pohranjeni unutar kolekcija, koje su ekvivalentne tablicama u relacijskim bazama podataka. Za razliku od tradicionalnih SQL baza podataka, gdje se podaci pohranjuju u strogo definiranim redovima i stupcima, MongoDB koristi Binary JSON (BSON) format za pohranu dokumenata. BSON je varijanta JavaScript Object Notation (JSON)-a koja podržava različite tipove podataka i omogućuje veću fleksibilnost pri upravljanju podacima.[22]

Jedna od glavnih prednosti MongoDB-a je njegova fleksibilnost u modeliranju podataka. Budući da ne zahtijeva unaprijed definirane sheme, korisnici mogu pohranjivati podatke u dokumentima s različitim strukturama unutar istih kolekcija. Ova karakteristika omogućava brzu prilagodbu i proširivanje modela podataka prema potrebama aplikacije bez potrebe za migracijama ili promjenama u strukturi baze podataka. Uz to, MongoDB podržava horizontalno i vertikalno skaliranje, što znači da se može prilagoditi povećanim zahtjevima za pohranom i obradom podataka jednostavnim dodavanjem novih resursa. Otpornost na greške sustava poboljšava se primjenom tehnike sharding. Sharding dijeli podatke na manje dijelove i distribuira ih među različitim serverima, što omogućava bolju brzinu rada i skalabilnost sustava,

dok istovremeno smanjuje rizik od prekida rada u slučaju kvara pojedinog servera. Također, MongoDB nudi snažne mogućnosti za indeksiranje i pretraživanje podataka, što poboljšava brzinu upita i učinkovitost pretraživanja.[22]

Postoji više pristupa povezivanja s MongoDB bazom podataka, od kojih su najpoznatiji MongoDB Atlas, lokalna instalacija MongoDB-a i korištenje Docker kontejnera. MongoDB Atlas je potpuno upravljana cloud usluga koja omogućava jednostavno postavljanje, skaliranje i upravljanje bazom podataka u oblaku. U ovoj aplikaciji korištena je lokalna instalacija MongoDB-a, koja pruža potpunu kontrolu nad konfiguracijom i sigurnošću, kao i mogućnost rada bez potrebe za stalnom internetskom vezom.

Za vizualno upravljanje podacima i jednostavnije interakcije korišten je MongoDB Compass, grafički alat koji olakšava pregledavanje i upravljanje bazom podataka (Slika 5.1) U lijevom dijelu sučelja nalazi se navigacijska traka koja prikazuje popis baza podataka dostupnih na lokalnom MongoDB poslužitelju (localhost:27017). Postoje baze podataka pod nazivima *admin*, *analyst*, *config*, *local* i *serviceProvider*. Važno je napomenuti da su baze podataka *admin*, *config* i *local* automatski generirane od strane MongoDB sustava i služe za interno upravljanje konfiguracijom, autentifikacijom i ostalim sistemskim funkcijama MongoDB-a. Desni dio sučelja prikazuje popis kolekcija ili popis dokumenata unutar kolekcije, omogućavajući korisniku brz pregled i interakciju s podacima. U navedenom primjeru prikazana je kolekcija *encrypted_masks* koja trenutno sadrži tri dokumenta, tj. tri JSON objekta sastavljenih od ključeva *_id*, *userId*, *__v* i *encryptedMask*. Ključ *_id* predstavlja jedinstveni identifikator svakog dokumenta i automatski ga generira MongoDB, *__v* obično služi za praćenje verzija dokumenata, *userId* predstavlja korisnički ID dok je *encryptedMask* enkriptirana maska. Osim navedenog, korisnik može vidjeti i pristupiti raznim opcijama poput pregleda agregacija, shema, indeksa te postavki validacije za svaku kolekciju. Također je moguće dodavati nove podatke, izvoziti postojeće, uređivati ili brisati dokumente unutar kolekcije koristeći odgovarajuće opcije dostupne unutar sučelja.



Slika 5.1 Sučelje za pregled baze podataka.

5.4 Ključni alati i knjižnice

Za izradu ove aplikacije korišteni su brojni alati i knjižnice koji omogućavaju izradu funkcionalne i sigurne aplikacije. Ovdje su opisani samo najvažniji, dok se puni popis svih korištenih alata i knjižnica može pronaći u *package-lock.json* datotekama unutar GitHub repozitorija, čija je poveznica navedena u poglavlju 7.

Na frontend dijelu aplikacije korištena je Shadcn UI knjižnica za stilizaciju komponenti, koja omogućava jednostavnu integraciju s Tailwind CSS-om te Zod za validaciju podataka unutar formi. Na backend dijelu, ključne knjižnice uključuju Bcrypt za sigurno hashiranje lozinki, Mongoose za modeliranje podataka i upravljanje MongoDB bazom podataka te Jsonwebtoken za generiranje JWT tokena. Knjižnice će biti detaljnije opisane u nastavku.

5.4.1 Shadcn UI

Shadcn UI je definiran kao "kolekcija višekratno upotrebljivih komponenti koje možemo kopirati i zalijepiti u naše aplikacije". Za razliku od drugih knjižnica kao što su Material UI ili Chakra UI, Shadcn UI se ne može instalirati kao ovisnost, nije dostupan niti se distribuira putem npm-a.[23] Umjesto toga, programeri mogu preuzeti izvorni kod svake komponente izravno u svoju kodnu bazu, čime se omogućuje veća kontrola i prilagodljivost, što ga čini jedinstvenim među sličnim alatima. U kontekstu ove aplikacije, Shadcn UI sam odabrala zbog svoje fleksibilnosti i estetski privlačnih komponenti, što omogućuje brzo prototipiranje i izradu prilagodljivih korisničkih sučelja. Iako ima određene nedostatke, njegove prednosti čine ga odličnim izborom za potrebe ove aplikacije.

Neke od brojnih prednosti uključuju lijepo dizajnirane komponente, širok raspon opcija za izradu složenih korisničkih sučelja, te jednostavnu prilagodbu putem Tailwind CSS-a. Široko je prihvaćen, što znači da postoji podrška zajednice i brojni resursi. Ipak, ima i nedostatke, poput potrebe za ručnom instalacijom komponenti, što može biti naporno, te povećanja veličine koda, što zahtijeva više održavanja. Također, za učinkovito korištenje potrebno je dobro poznavanje Tailwind CSS-a, što može biti izazov za manje iskusne developere.[24]

5.4.2 Zod

Zod je knjižnica koja nadopunjuje TypeScript omogućujući provjeru tipova podataka u runtimeu, čime osigurava integritet podataka i sprječava unos nevažećih vrijednosti, poput brojeva umjesto očekivanih stringova. Ova mogućnost je ključna za aplikacije koje obrađuju korisničke unose, kao što su obrasci, gdje TypeScript sam po sebi ne može osigurati valjanost podataka tijekom izvođenja. Zod, za razliku od drugih alata poput Yup ili Joi, omogućuje fleksibilno definiranje shema bez vanjskih ovisnosti te se ističe svojom nepromjenjivošću i kompatibilnošću s TypeScriptom, čime postaje superioran izbor za izradu robusnih aplikacija.[25]

Zod knjižnicu koristila sam unutar aplikacije za definiranje shema formi za login, registraciju i unos podataka o frekvenciji pojavljivanja kibernetičkih napada. Slje-

deći dio programskog koda prikazuje primjer korištenja Zod knjižnice za definiranje sheme registracijske forme unutar aplikacije. Definira se objekt formSchema koji sadrži pravila validacije za polja kao ime, email i lozinka. Email polje provjerava valjanost formata, dok lozinka mora imati minimalno 8 znakova. Polje za ime zahtijeva da sadrži najmanje 2, a najviše 30 znakova te smije sadržavati samo slova. Ova shema omogućava provjeru ispravnosti korisničkih podataka prije nego što se pošalju na poslužitelj, osiguravajući integritet podataka i bolje korisničko iskustvo. Korištenje Zoda na ovaj način pomaže smanjiti pogreške pri unosu i poboljšava sigurnost aplikacije.

```
import { z } from "zod";

const formSchema = z.object({
  email: z.string().email({
    message: "Please enter a valid email.",
  }),
  password: z.string().min(8, {
    message: "Password must be at least 8 characters.",
  }),
  name: z
    .string()
    .min(1, { message: "Name is required." })
    .min(2, { message: "Name must be at least 2 characters long
       ." })
    .max(30, { message: "Name must not exceed 30 characters."
       })
    .regex(/^[A-Za-z]+$/, { message: "Name can only contain
       letters." }),
});

export default formSchema;
```

5.4.3 Bcrypt

Bcrypt je popularna knjižnica za Node.js koja se koristi za sigurno hashiranje lozinki, pružajući efikasan način za zaštitu korisničkih podataka. Ova knjižnica primjenjuje algoritam bcrypt koji je dizajniran da bude spor i otporan na napade brute-force metodom. Algoritam su dizajnirali Niels Provos i David Mazières, a temelji se na algoritmu Blowfish. Bcrypt uključuje "salt" – nasumično generirani niz koji se dodaje lozinci prije hashiranja – čime se dodatno otežava napad s predračunatim hash tablicama. Osim toga, bcrypt omogućuje konfiguriranje broja radnih iteracija, što omogućava povećanje računalnih resursa potrebnih za hashiranje lozinke i time povećava sigurnost podataka.[26]

Sljedeća linija koda nalazi se unutar login kontrolera te koristi bcrypt knjižnicu za hashiranje lozinke. Konkretno, *bcrypt.hash(password, saltRounds)* je funkcija koja uzima lozinku i broj *saltRounds*, koji u ovom slučaju iznosi 10. *saltRounds* označava broj puta koliko će algoritam koristiti "salt" za dodavanje slučajnosti u proces hashiranja. Rezultat ove funkcije je hashirana verzija lozinke koja se pohranjuje u varijablu *hashedPassword*. Upotreba *await* znači da će izvršenje funkcije čekati da se proces hashiranja završi prije nego što nastavi, omogućujući time pravilno rukovanje s asinkronim operacijama.

```
const hashedPassword = await bcrypt.hash(password, saltRounds);
```

U kontroleru za registraciju, sljedeća linija koda koristi bcrypt knjižnicu za usporedbu lozinke unesene od strane korisnika s hashiranom lozinkom pohranjenom u bazi podataka. Funkcija *bcrypt.compare* prihvaća dva argumenta: password, što je lozinka koju korisnik unosi prilikom prijave, i dbUser.password, što je hashirana lozinka pohranjena u bazi podataka. Funkcija vraća true ako unesena lozinka odgovara hashiranoj lozinki, a false ako se ne podudaraju.

```
const match = await bcrypt.compare(password, dbUser.password);
```

5.4.4 Mongoose

Mongoose je popularna knjižnica za Node.js koja omogućuje elegantno modeliranje podataka u MongoDB, pružajući jednostavan način za definiranje shema i valida-

ciju podataka. Korištenjem ove knjižnice, objekti u kodu lako se prevode u njihovu odgovarajuću reprezentaciju u MongoDB, što značajno pojednostavljuje upravljanje podacima unutar aplikacija. Podrška za napredne značajke poput middleware-a, virtualnih atributa i query builders-a dodatno omogućuje pisanje modularnog i održivog koda, čineći razvoj s MongoDB-om još fleksibilnijim i učinkovitijim.[27]

Sljedeći primjer pokazuje korištenje Mongoose knjižnice u aplikaciji za definiranje modela korisnika (*User*). Kod uključuje shemu *userSchema* koja specificira polja *name*, *email* i *password*, zajedno s njihovim pravilima validacije. Sva polja su obavezna, tj. neophodna su za stvaranje korisnika u bazi podataka. Također, za polje email koristi se knjižnica validator kako bi se osiguralo da je unijeti email ispravan, dok polje password mora imati minimalnu dužinu od 8 znakova. Iako je validacija podataka već implementirana na frontend strani pomoću Zod knjižnice, ova backend validacija je uključena kao dodatna sigurnosna mjera kako bi se osiguralo da svi podaci koji ulaze u bazu podataka budu validni i konzistentni.

```
import mongoose from "mongoose";
import isEmail from "validator/lib/isEmail.js";

const userSchema = new mongoose.Schema({
  name: {
    type: String,
    required: [true, "Please enter a name"],
  },
  email: {
    type: String,
    required: [true, "Please enter email"],
    unique: true,
    validate: [isEmail, "Please enter a valid email"],
  },
  password: {
    type: String,
    required: [true, "Please enter password"],
    minLength: [6, "Minimum length is 6 characters"],
  },
```

```
});

const User = mongoose.model("user", userSchema);

export default User;
```

5.4.5 Jsonwebtoken

Jsonwebtoken je Node.js knjižnica koja pruža funkcionalnost kreiranja i verificiranja JWT-a, čime se olakšava upravljanje autentifikacijom i autorizacijom korisnika u modernim web aplikacijama. Knjižnica podržava različite algoritme za potpisivanje tokena, uključujući HMAC i RSA te omogućuje konfiguriranje opcija poput vremena isteka tokena, što je ključno za osiguranje da tokeni ne postanu dugotrajna sigurnosna prijetnja. Ova knjižnica također nudi fleksibilnost u definiranju korisničkih podataka unutar tokena, omogućujući prijenos specifičnih informacija koje su bitne za aplikaciju, kao što su korisnički ID, uloge ili dozvole.[28]

Navedeni isječak programskog koda nalazi se unutar login kontrolera te koristi jsonwebtoken knjižnicu za stvaranje JWT-a. U njemu se generira token koji uključuje korisnikove podatke: _id, name, i email. Ovi podaci su pohranjeni unutar tokena kao dio njegovog payloada, čime se korisnik može identificirati u budućim zahtjevima bez potrebe za stalnim pristupom bazi podataka. Drugi argument *jwt.sign* metode je tajni ključ (JWT_SECRET_KEY), koji je pohranjen unutar vrijednosti okruženja *process.env*, a koristi se za digitalno potpisivanje tokena, osiguravajući njegovu sigurnost i integritet. Treći argument postavlja vrijeme isteka tokena na jedan dan ("1d"), nakon čega će token postati nevažeći i korisnik će se morati ponovo prijaviti u aplikaciju.

```
const jwtToken = jwt.sign(
          { _id: dbUser._id, name: dbUser.name, email },
          process.env.JWT_SECRET_KEY,
          { expiresIn: "1d" }
        );
```

Poglavlje 6

Analiza programskog koda

Ovo poglavlje obuhvaća analizu programskog koda koji sadrži veliki broj linija i složenih funkcionalnosti. Zbog opsežnosti koda, fokusirat ću se na ključne dijelove koji su relevantni za implementaciju MPC-a, pružajući detaljna objašnjenja kako bi se razumjela osnovna logika i struktura koja stoji iza navedenih dijelova koda. Prije analize programskog koda prikazat ću i objasniti sekvencijalni dijagram koji će vizualno prikazati tijek izvršenja i interakcije među ključnim komponentama.

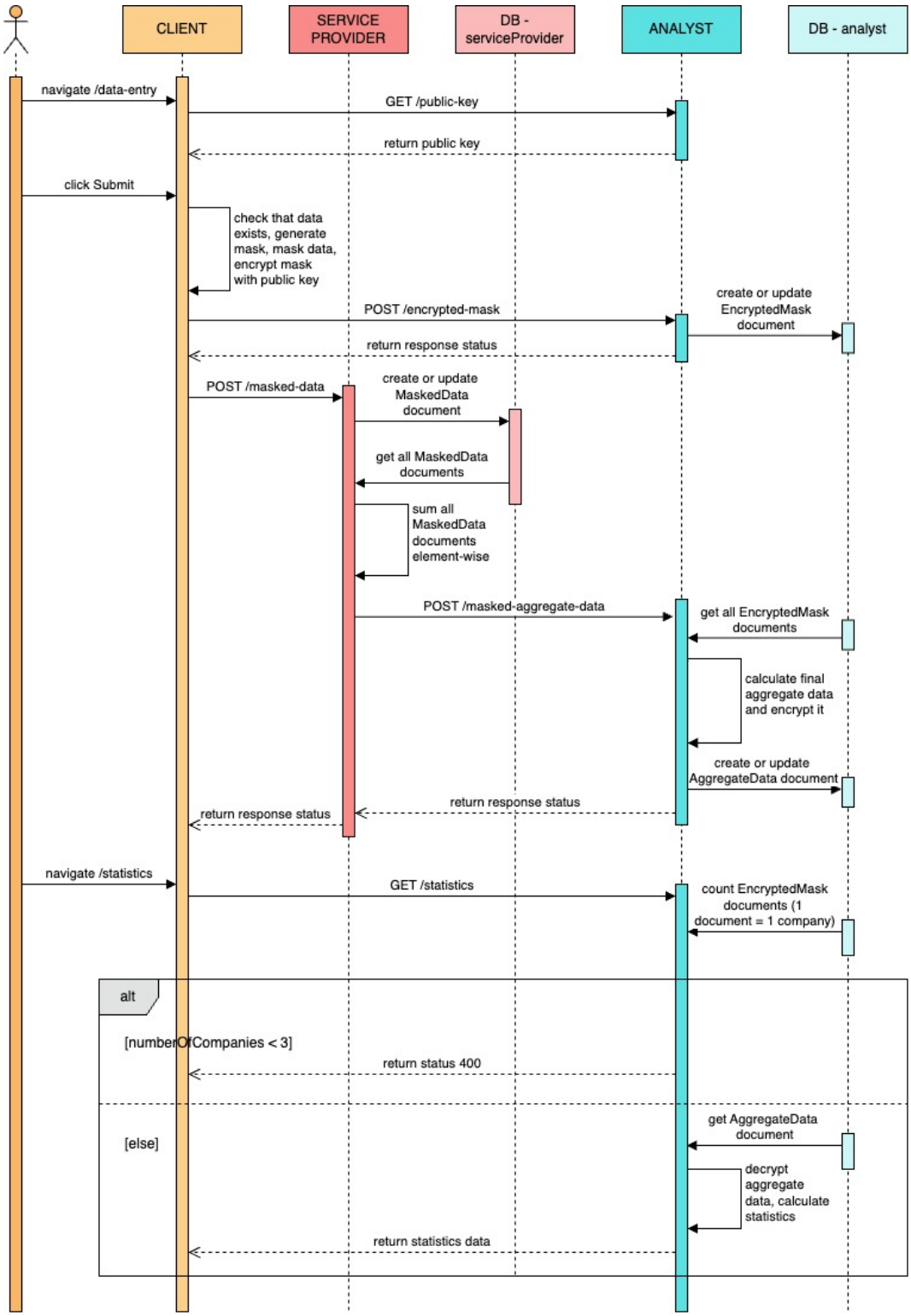
6.1 Dijagram sekvenci

Dijagrami sekvenci se koriste za modeliranje interakcija između različitih objekata u sustavu tijekom vremena. Ovi dijagrami sadrže osnovne elemente poput objekata koji sudjeluju u komunikaciji i njihovih "životnih linija", koje prikazuju vremenski tijek postojanja objekata i njihovu uključenost u razmjenu poruka. Poruke su prikazane strelicama koje pokazuju komunikaciju ili protok informacija među objektima, a mogu biti različitih tipova, poput sinhronih, asinhronih ili povratnih poruka. Dijagrami sekvenci su iznimno korisni tijekom faze dizajniranja softvera jer omogućuju detaljno razumijevanje zahtjeva sustava i njegove funkcionalnosti, a također pomažu u dokumentiranju složenih procesa unutar sustava.

Slika 6.1 prikazuje dijagram sekvenci ove aplikacije. Komunikacija se odvija između šest objekata, to jest entiteta: korsnik, klijent, davatelj usluga, baza podataka

davatelja usluga, analitičar i baza podataka analitičara. Svaki od objekata ima svoju "životnu liniju" karakterističnе boje, dok su poruke označene crnim punim ili isprekidanim strelicama. Pri dnu dijagrama nalazi se tzv. alt okvir, koji označava if-else petlju. Drugim riječima, okvir je podijeljen na dva dijela, od kojih se samo jedan može izvršiti ovisno o uvjetu. U ovome slučaju uvjet je da je broj tvrtki manji od tri. Ako je uvjet točan, analitičar vraća status 400 i ne generiraju se statistike. Suprotno, ako je broj tvrtki veći ili jednak tri, statistike se generiraju i šalju klijentu.

U ovom dijagramu prikazan je proces obrade podataka od trenutka kada korisnik otvori stranicu za unos podataka (*/data-entry*) do trenutka kada klijent dobije finalne statistike. Cijeli proces detaljno će biti objašnjen u sljedećem potpoglavlju, gdje će poruke između objekata biti potkrijepljene dijelovima programskog koda. Budući da su svi objekti uključeni u komunikaciju u više navrata, slijed izvršenja programskog koda bit će objašnjen kronološki radi lakšeg razumijevanja cijelog procesa.



Slika 6.1 Dijagram sekvenci.

6.2 Kronološki slijed izvršenja programskog koda

Kada korisnik otvori stranicu za unos podataka, automatski se pokreće proces dohvaćanja javnog ključa sa poslužitelja analitičara. Navedena funkcija *fetchPublicKey* inicira HTTP zahtjev prema serveru koristeći GET metodu. Nakon što poslužitelj obradi zahtjev, funkcija prima odgovor koji sadrži podatke u JSON formatu. Ako je zahtjev uspješan, javni ključ se preuzima iz odgovora i pohranjuje ga pomoću *useState* kuke, koja ga čini dostupnim ostatku aplikacije za daljnju upotrebu. U slučaju da dođe do greške tijekom dohvaćanja javnog ključa, funkcija generira grešku na web sučelju i ispisuje detalje pogreške u terminal.

```
const fetchPublicKey = async () => {
    try {
        const response = await fetch(
            `${process.env.NEXT_PUBLIC_ANALYST_URL}/public-key
              `,
            {
              method: "GET",
              headers: {
                "Content-type": "application/json",
              },
            }
        );

        const responseData = await response.json();
        if (!response.ok) {
            toast.error(responseData.message);
        }
        const fetchedPublicKey = responseData.publicKey;
        setPublicKey(fetchedPublicKey);
    } catch (error) {
        console.log(error);
        throw new Error("Cannot fetch public key.");
    }
};
```

U cijeloj aplikaciji sintaksa slanja zahtjeva putem *fetch* funkcije dosljedno je strukturirana, pri čemu se mijenjaju samo ključni parametri ovisno o specifičnom zahtjevu. Za svaku situaciju potrebno je prilagoditi HTTP metodu (GET ili POST), odrediti hoće li se slati podaci u tijelu zahtjeva te specificirati odgovarajuću adresu poslužitelja na koji se zahtjev šalje. Ovakva konzistentna sintaksa omogućava jednostavniju prilagodbu i održavanje koda, jer su sve promjene usmjerene na ove tri ključne varijable.

Pritiskom gumba *Submit* na stranici za unos podataka, poziva se asinkrona funkcija *onSubmit*. Prvo, funkcija provjerava jesu li svi unosi u obrascu nule, što bi značilo da korisnik nije unio valjane podatke. Ako su svi podaci nule, funkcija prikazuje poruku o pogrešci i zaustavlja daljnje izvršavanje. Ako nisu svi podaci nule, funkcija generira slučajnu numeričku masku između -1000000 i 1000000, koja služi za skrivanje stvarnih podataka. Ova maska se enkriptira korištenjem ranije dohvaćenog javnog ključa, čime se dodatno osigurava njezina sigurnost. Svaki podatak u obrascu se zatim "maskira" tako da se vrijednostima u obrascu dodaje iznos slučajne maske. Funkcija zatim šalje korisnički ID i enkriptiranu masku poslužitelju analitičaru na endpoint */api/encrypted-mask*. Poslužitelj analitičar pohranjuje dobivene podatke unutar *encrypted_masks* kolekcije u bazi podataka te ukoliko već postoji dokument sa jednakim korisničkim ID-jem, dokument se samo ažurira. Nakon toga maskirani podaci se zajedno sa korisničkim ID-jem šalju poslužitelju davatelju usluga na endpoint */api/masked-data* putem HTTP POST zahtjeva. Ako bilo koji od ovih koraka ne uspije, prikazuje se poruka o pogrešci, a ako su svi koraci uspješno završeni, korisnik dobiva obavijest da su njegovi podaci uspješno poslani.

Sljedeći kod predstavlja kontroler koji obrađuje zahtjev poslan sa klijenta na endpoint */api/masked-data* poslužitelja davatelja usluga. Kada klijent pošalje POST zahtjev s maskiranim podacima, ovaj kontroler preuzima podatke iz tijela zahtjeva, a zatim ih pohranjuje u bazu podataka (kolekcija *masked_data*). Konkretno, kod koristi metodu *findOneAndUpdate* kako bi pronašao zapis u bazi podataka prema korisničkom ID-ju i ažurirao podatke. Ako zapis ne postoji, kreira se novi (zahvaljujući opciji *upsert*). U slučaju greške prilikom spremanja podataka, kontroler vraća status greške 500 i odgovarajuću poruku. Ako su podaci uspješno spremljeni, kontroler poziva funkciju *sendMaskedAggregateDataToAnalyst*.

```
export const maskedDataController = async (req, res) => {
  try {
    const { userId, data } = req.body;

    await MaskedData.findOneAndUpdate(
      { userId },
      { data },
      { upsert: true, new: true, setDefaultsOnInsert: true }
    );

    const analystResult = await
       sendMaskedAggregateDataToAnalyst();

    if (analystResult.success) {
      res
        .status(200)
        .json({ message: "Data saved and sent to analyst
           successfully" });
    } else {
      res.status(500).json({
        error: "Data saved, but failed to send to analyst",
        details: analystResult.error,
      });
    }
  } catch (error) {
    console.error("Error saving data:", error);
    res.status(500).json({ error: "An error occurred while
       saving the data" });
  }
};
```

Funkcija *sendMaskedAggregateDataToAnalyst* najprije dohvaća sve dokumente iz kolekcije *masked_data*, pri čemu iz svakog dokumenta uzima samo polje *data*, koje sadrži niz podataka. Zatim se inicijalizira 10x12 dvodimenzionalni niz ispunjen nulama, koji će služiti kao spremnik za agregiranje podataka. Svaki dokument se

prolazi redak po redak te se vrijednosti zbrajaju u odgovarajuće ćelije agregatnog niza. Nakon agregacije, rezultirajući podaci se pretvaraju u JSON format i šalju putem HTTP POST zahtjeva na endpoint */api/masked-aggregate-data* poslužitelja analitičara. Ako slanje prođe uspješno, ispisuje se potvrda, a eventualne greške se bilježe za daljnju analizu.

```
const sendMaskedAggregateDataToAnalyst = async () => {
  try {
    const allMaskedData = await MaskedData.find({}, "data");

    const maskedAggregateData = Array.from({ length: 10 }, ()
       =>
      Array(12).fill(0)
    );

    allMaskedData.forEach((record) => {
      record.data.forEach((row, rowIndex) => {
        row.forEach((cell, cellIndex) => {
          if (rowIndex < 10 && cellIndex < 12) {
            maskedAggregateData[rowIndex][cellIndex] += cell;
          }
        });
      });
    });

    const response = await fetch(
      `${process.env.ANALYST_URL}/masked-aggregate-data`,
      {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          "Access-Control-Allow-Origin": "*",
        },
        body: JSON.stringify({ maskedAggregateData:
           maskedAggregateData }),
```

```
      }
    );

    if (!response.ok) {
      throw new Error(`Analyst responded with status: ${
         response.status}`);
    }

    console.log("Masked aggregate data sent to analyst
       successfully");
    return { success: true };
  } catch (error) {
    console.error("Error sending masked data to analyst:",
       error);
    return { success: false, error: error.message };
  }
};
```

Sljedeći kod predstavlja kontroler poslužitelja analitičara *maskedAggregateDataController* koji prima zahtjev poslan sa poslužitelja davatelja usluga. Kontroler prvo provjerava postoje li maskirani agregatni podaci u tijelu zahtjeva. Ako podaci nisu prisutni, vraća HTTP status 400 s porukom o grešci. Ako podaci postoje, kontroler dohvaća agregatnu masku putem *getAggregateMask* funkcije, koja uzima sve enkriptirane maske spremljene u *encrypted_masks* kolekciju baze podataka, dekriptira ih te vraća njihov zbroj. Nakon toga, kontroler od svakog elementa u *maskedAggregateData* oduzima tu masku i tako dobiva finalne agregatne podatke. Finalni agregatni podaci moraju se enkriptirati prije spremanja u bazu podataka kako bi se zaštitila privatnost podataka prve tvrtke. Podaci su enkriptirani pomoću algoritma aes-256-ecb i simetričnog 32-bajtnog ključa. Enkriptirani agregatni podaci se zatim spremaju u kolekciju *encrypted_aggregate_data* u bazi podataka koristeći metodu *findOneAndUpdate* te se vraća HTTP status 200 s porukom o uspjehu. Ako dođe do greške, vraća se HTTP status 500 s odgovarajućom porukom o grešci.

```
export const maskedAggregateDataController = async (req, res)
   => {
```

```
  try {
    const { maskedAggregateData } = req.body;

    if (!maskedAggregateData) {
      return res
        .status(400)
        .json({ error: "No masked aggregate data provided" });
    }

    const aggregateMask = await getAggregateMask();

    const aggregateData = maskedAggregateData.map((row) =>
      row.map((value) => value - aggregateMask)
    );

    const encryptedAggregateData = encryptAggregateData(
       aggregateData);

    await AggregateData.findOneAndUpdate(
      {},
      { data: encryptedAggregateData },
      { upsert: true, new: true, setDefaultsOnInsert: true }
    );

    res.status(200).json({ message: "Processed aggregate data!"
        });
  } catch (error) {
    console.error("Error processing masked aggregate data:",
       error);
    res.status(500).json({ error: "Internal Server Error" });
  }
};
```

Pohranjivanjem enkriptiranih agregatnih podataka završava backend proces pokrenut pritiskom na gumb *Submit* na stranici za unos podataka.

Navigirajući na stranicu */statistics*, pokreće se HTTP GET zahtjev prema poslužitelju analitičaru na endpoint */api/statistics*. Kontroler zaslužan za obrađivanje tog zahtjeva broji ukupan broj dokumenata u kolekciji *encrypted_masks* kako bi provjerio ima li dovoljno podataka za analizu, vraćajući status 400 ako je manje od tri tvrtke poslalo podatke. Ukoliko je uvjet zadovoljen, kontroler pokušava pronaći dokument spremljen unutar kolekcije *encrypted_aggregate_data*. Ako dokument nije pronađen, kontroler vraća HTTP status 400 s porukom o grešci. Ako dokument postoji, njegov sadržaj se dekriptira pomoću funkcije *decryptAggregateData*.

Nakon što su ispunjeni svi uvjeti za analizu podataka, kontroler prelazi na obradu statistike. Prvo, inicijalizira tri varijable: *totalAttacks* za ukupan broj napada, *monthlyAttacks* za broj napada po mjesecima i *attackTypeCounts* za broj napada po tipu. Ove varijable služe za prikupljanje i sumiranje podataka iz dvodimenzionalnog niza *aggregateData*, gdje su redovi tipovi napada, a stupci predstavljaju mjesece. Funkcija prolazi kroz svaki element niza i ažurira ove varijable dodajući broj napada u odgovarajuće statistike. Nakon što su svi podaci obrađeni, koristi se *Math.max* i *Math.min* za pronalaženje najčešćih i najrjeđih tipova napada.

Zatim, podaci se formatiraju kako bi odgovarali potrebnom formatu za generiranje dijagrama na frontendu. Specifično, za svaki mjesec se kreira objekt koji uključuje naziv mjeseca i broj napada za svaki tip napada u tom mjesecu. Na kraju, sastavlja se odgovor koji uključuje broj tvrtki, ukupni broj napada, najčešće i najmanje česte napade te formatirane podatke o napadima, te šalje HTTP status 200 s tim informacijama u JSON formatu. Ako dođe do greške tijekom obrade, funkcija bilježi grešku u konzolu i vraća HTTP status 500. Ovisno o odgovoru koji dobije od poslužitelja analitičara, klijent može prikazati dijagrame sa statistikom, karticu s obavijesti da nema dovoljno podataka ili obavijest o pogrešci. Time završava proces implementacije MPC-a unutar aplikacije.

```
export const statisticsController = async (req, res) => {
  try {
    const numberOfCompanies = await EncryptedMask.
       countDocuments();

    if (numberOfCompanies < 3) {
```

```
    return res
      .status(400)
      .json({ message: "Not enough data entries for analysis.
        " });
  }

  const aggregateDataEntry = await AggregateData.findOne({});

  if (!aggregateDataEntry) {
    return res.status(400).json({ message: "No aggregate data
        found" });
  }

  const data = aggregateDataEntry.data;
  const aggregateData = decryptAggregateData(data);

  let totalAttacks = 0;
  let monthlyAttacks = new Array(12).fill(0);
  let attackTypeCounts = new Array(10).fill(0);

  aggregateData.forEach((attackTypeArray, attackTypeIndex) =>
      {
    attackTypeArray.forEach((monthlyCount, monthIndex) => {
      totalAttacks += monthlyCount;
      monthlyAttacks[monthIndex] += monthlyCount;
      attackTypeCounts[attackTypeIndex] += monthlyCount;
    });
  });

  const maxAttackTypeCount = Math.max(...attackTypeCounts);
  const minAttackTypeCount = Math.min(...attackTypeCounts);

  const mostCommonAttack = attackTypes.filter(
    (_, index) => attackTypeCounts[index] ===
      maxAttackTypeCount
```

```
    );
    const leastCommonAttack = attackTypes.filter(
      (_, index) => attackTypeCounts[index] ===
         minAttackTypeCount
    );

    const formattedAggregateData = months.map((month, index) =>
       {
      const monthData = { month };
      attackTypes.forEach((attackType, attackIndex) => {
        monthData[attackType] = aggregateData[attackIndex][
           index];
      });
      return monthData;
    });

    const response = {
      companies: numberOfCompanies,
      totalAttacks,
      mostCommonAttack:
        mostCommonAttack.length === 1 ? mostCommonAttack[0] :
           mostCommonAttack,
      leastCommonAttack:
        leastCommonAttack.length === 1
          ? leastCommonAttack[0]
          : leastCommonAttack,
      aggregateData: formattedAggregateData,
    };

    res.status(200).json(response);
  } catch (error) {
    res.status(500).json({ message: "Error generating
       statistics." });
  }
};
```

Poglavlje 7

Testiranje aplikacije

Izvorni kod aplikacije može se pronaći na Github repozitoriju, dostupnom na poveznici `https://github.com/antoniagrabar/mpc-fullstack-app`. Kako bi se repozitorij klonirao na lokalni disk, potrebno je pozicionirati se unutar željene mape na lokalnom disku te u terminal upisati sljedeću naredbu:

```
git clone https://github.com/antoniagrabar/mpc-fullstack-app.
    git
```

Za ovaj postupak preporučuje se korištenje Visual Studio Code (VSC) editora zbog svoje integrirane Git funkcionalnosti i jednostavnog pregleda preuzetog koda. Programski kod je strukturiran u tri mape: *client*, *server-service-provider* i *server-analyst* koji redom označuju klijentsku aplikaciju, poslužitelj davatelj usluga te poslužitelj analitičar.

Nakon preuzimanja programskog koda, potrebno je instalirati sve potrebne alate i knjižnice potrebne za uspješno pokretanje aplikacije. Sve zavisnosti, kao i njihove verzije te informacije o projektu su zapisane unutar pojedinačnih package.json datoteka. Iz izvorne lokacije potrebno je ući u svaku od tri mape i izvršiti sljedeću naredbu:

```
npm install
```

Pokretanjem navedene naredbe preuzimaju se svi potrebni paketi i njihove zavisnosti s interneta, a zatim se pohranjuju u mapu node_modules. Na taj način

aplikaciji se omogućuje korištenje potrebnih vanjskih knjižnica i alata.

Kao što je već spomenuto, MongoDB se može povezati na više načina, od kojih su najpoznatiji MongoDB Atlas i lokalna instalacija. Za ovu aplikaciju odabrala sam lokalnu instalaciju kako bi dodatno očuvala sigurnost podataka ne spremajući ih u oblak. Za lokalnu instalaciju potrebno je najprije pratiti upute sa poveznice `https://www.mongodb.com/docs/manual/installation/`, ovisno o korištenom operacijskom sustavu. Verzija koju sam koristila je *MongoDB Community Edition* za MacOS. Nakon instalacije bitno je pokrenuti MongoDB, u mome slučaju trebala sam pokrenuti sljedeću naredbu u terminalu:

```
brew services start mongodb-community@7.0
```

Za pregled baze podataka koristila sam MongoDB Compass GUI, koji je preuzet sa poveznice `https://www.mongodb.com/try/download/compass`. Nakon preuzimanja potrebno je stvoriti novu vezu tako da se opcionalno uredi već zadani URL te klikne na gumb "Save & Connect". Port 27017 je unaprijed postavljen kao zadani port za MongoDB u konfiguraciji te na ovom portu MongoDB server prima dolazne veze od klijenata i aplikacija koje žele pristupiti podacima, izvršavati upite i manipulirati podacima. Budući da je na jednom uređaju moguće spajanje samo na jedan port, baze podataka poslužitelja davatelja usluga i poslužitelja analitičara su smještene unutar iste MongoDB veze. Naravno, ukoliko bi se aplikacije poslužitelja pokretale na fizičkim poslužiteljima, tada bi svaki poslužitelj imao potpuno fizički odvojenu bazu podataka kako bi se 100% osiguralo da davatelj usluga i analitičar ne mogu doći do međusobnih baza podataka.

Nakon instalacije MongoDB-a i stvaranja nove veze, potrebno je u VSC-u unutar mape *server-service-provider* stvoriti *.env.local* datoteku te u nju najprije kopirati sadržaj iz *.env-example* datoteke. Budući da *.env* datoteke obično sadrže konfiguracijske podatke i osjetljive informacije, one se ne bi trebale javno objaviti na Githubu. Iz tog razloga kreirala sam *.env-example* datoteku koja sadrži imena varijabli koje *.env.local* datoteka treba sadržavati za ispravan rad aplikacije. Jedna od varijabli je MONGO_URI, koja se treba postaviti na generirani MongoDB URL. Na promijenjeni ili zadani URL "mongodb://localhost:27017" potrebno je dodati "/{naziv_baze}", gdje je {naziv_baze} proizvoljan string. Za poslužitelja davate-

lja usluga odabrala sam naziv *serviceProvider*, dok sam za poslužitelja analitičara odabrala *analyst*. Isti postupak kreiranja *.env.local* datoteke i postavljanja varijable MONGO_URI treba se ponoviti unutar mape *server-analyst*.

U *.env-example* datoteci unutar mape *server-service-provider*, nedostaju još tri varijable: PORT, JWT_SECRET_KEY i ANALYST_URL. PORT je varijabla koja definira port poslužitelja. Broj porta je u potpunosti proizvoljan, no bitno je da nije jednak portu klijentske aplikacije. Neki od uobičajenih portova su 3000, 8080, 80, 5000 i 443. ANALYST_URL je adresa koja se koristi za pristup API endpointu poslužitelja analitičara i obično je oblika "http://localhost:{port}/api". Zadnja varijabla je JWT_SECRET_KEY, što predstavlja tajni ključ koji se koristi za stvaranje digitalnog potpisa za JWT. Tajni ključ je također proizvoljan, no trebao bi biti dovoljno složen i nepredvidljiv kako bi spriječio napadače da ga pogode ili otkriju brute-force metodom. JWT_SECRET_KEY sam generirala upisivanjem sljedeće naredbe u terminal:

```
node -e "console.log(require('crypto').randomBytes(256).
    toString('base64'));
```

Ovaj kod generira slučajni niz od 256 bajtova, koji se zatim pretvara u Base64 format i prikazuje u terminalu. Niz se zatim dodaje u *.env.local*. Unutar *.env.local* datoteke mape *server-analyst* nedostaju još samo varijable PORT i SYMMETRIC_KEY. Za varijablu PORT vrijede jednaka pravila kao i ranije. Naravno, port treba biti drugačiji od porta poslužitelja davatelja usluga. Varijabla SYMMETRIC_KEY označava 32-bajtni simetrični ključ u heksadecimalnom formatu koji je potreban za enkripciju i dekripciju agregatne statistike. Ključ je proizvoljan, no bitno je da je duljina ključa točno 32 bajta kako bi algoritam ispravno radio. Ključ se može generirati upisivanjem sljedećeg koda u terminal, nakon čega ga je potrebno kopirati u *.env.local* datoteku.

```
node -e "const crypto = require('crypto'); console.log(crypto.
    randomBytes(32).toString('hex'));"
```

U mapi *client* također je potrebno stvoriti *.env.local* datoteku sa varijablama iz *.env-example*. Prva varijabla je NEXT_PUBLIC_SERVICE_PROVIDER_URL te ona predstavlja adresu koja se koristi za pristup API endpointu poslužitelja da-

vatelja usluga. Jednako tako, varijabla NEXT_PUBLIC_ANALYST_URL se koristi za pristup API endpointu poslužitelja analitičara. Obje varijable trebaju biti oblika "http://localhost:{port}/api", gdje port predstavlja pojedinačni port svakog poslužitelja. NEXTAUTH_URL varijabla treba se postaviti na lokalnu adresu na kojoj se pokreće klijentska aplikacija. NEXTAUTH_SECRET je tajni ključ potreban za ispravan rad NextAuth.js knjižnice i treba se podudarati sa tajnim ključem JWT_SECRET_KEY poslužitelja davatelja usluga. Zadnja varijabla je NEXT_PUBLIC_ANALYST_EMAIL, koja je proizvoljna i postavlja se na email sa kojim će se prijavljivati analitičar u sustav. Analitičar ima pristup isključivo statistikama i ne može poslati podatke.

Sljedeće, potrebno je u terminal unutar *server-analyst* mape upisati sljedeće dvije naredbe. Prva naredba generira RSA privatni ključ od 2048 bita i sprema ga u datoteku *keypair.pem*. Druga naredba izvlači javni ključ iz privatnog ključa u datoteci *keypair.pem* i sprema ga u datoteku *publickey.crt*.

```
openssl genrsa -out keypair.pem 2048
openssl rsa -in keypair.pem -pubout -out publickey.crt
```

Nakon uspješno provedenih svih prethodnih koraka, aplikacija se može uspješno pokrenuti. Potrebno je otvoriti tri terminala unutar VSC-a, od kojih je svaki pozicioniran unutar jedne od tri glavnih mapa. U svaki terminal je potrebno upisati sljedeće:

```
npm run dev
```

Klijentska aplikacija otvara se u željenom web pregledniku upisujući URL adresu koja je navedena odmah nakon izvođenja gornje naredbe. Zadana URL adresa je obično "http://localhost:3000". Budući da kod prvog pokretanja aplikacije korisnik još nije prijavljen, apilikacija će se automatski preusmjeriti na stranicu za prijavu.

Korisnik koji nema jos račun mora se najprije registrirati. To može učiniti klikom na poveznicu "Don't have an account? Register here" ili upisivanjem URL-a "http://localhost:3000/register" u adresnu traku. Stranica za registraciju prikazana je na Slici 7.1. Kako bi se registrirao, korisnik treba upisati svoje ime, email i lozinku vodeći računa o ispravnim formatima koje zahtjeva aplikacija. Ime treba imati između 2 i 30 znakova koji su isključivo slova, email treba biti u standardnom formatu

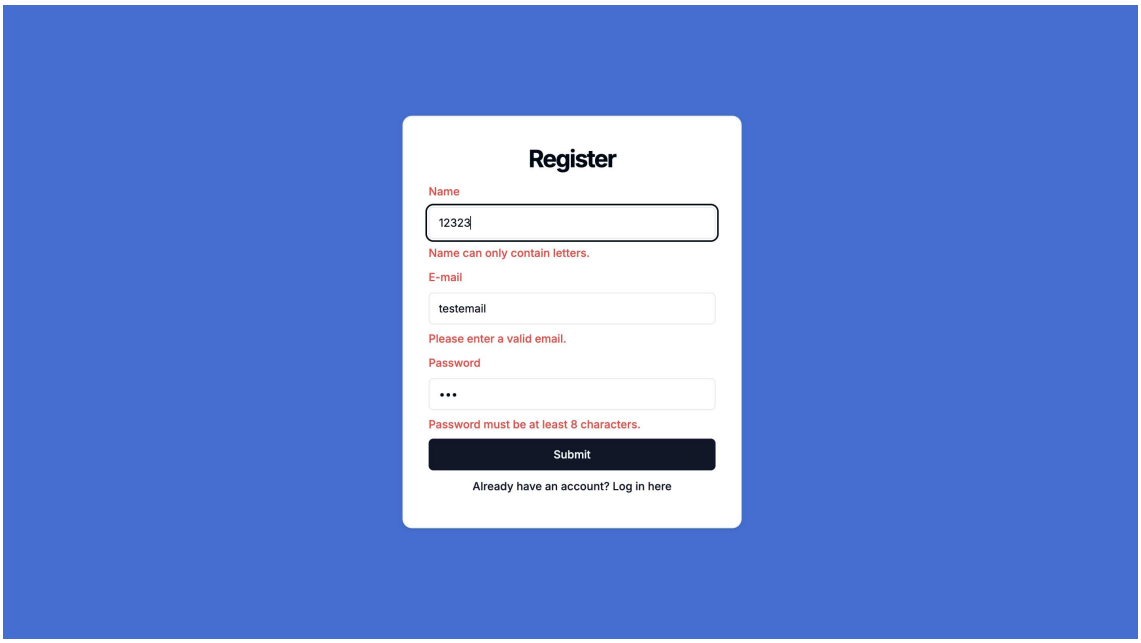
emaila, dok lozinka treba sadržavati minimalno 8 znakova. Ukoliko je registracija uspješna, pokazuje se kartica sa Slike 7.2. Potrebno je kliknuti na gumb "Login" kako bi se vratila stranica za prijavu, ili upisati "http://localhost:3000/login" u adresnu traku. Proces prijave je uobičajen i zahtijeva od korisnika da upiše svoj email i lozinku sa kojom se registrirao, poštujući ista validacijska pravila.

Uspješno prijavljenog korisnika aplikacija preusmjeruje na početnu stranicu prikazanu na Slici 7.4. Sučelje je vrlo jednostavno i intuitivno. U glavnom dijelu stranice nalaze se kratke upute za unos podataka. Naglasak je stavljen na privatnost i sigurnost podataka, a pojašnjava se kako aplikacija koristi MPC kako bi osigurala da podaci ostanu potpuno skriveni. S lijeve strane sučelja nalazi se jednostavna navigacija koja sadrži tri opcije: "Home" za trenutnu stranicu sa uputama, "Data Entry" za unos podataka te "Statistics" za pregled rezultata analize. Na dnu navigacije nalazi se gumb "Log Out" za odjavu iz aplikacije.

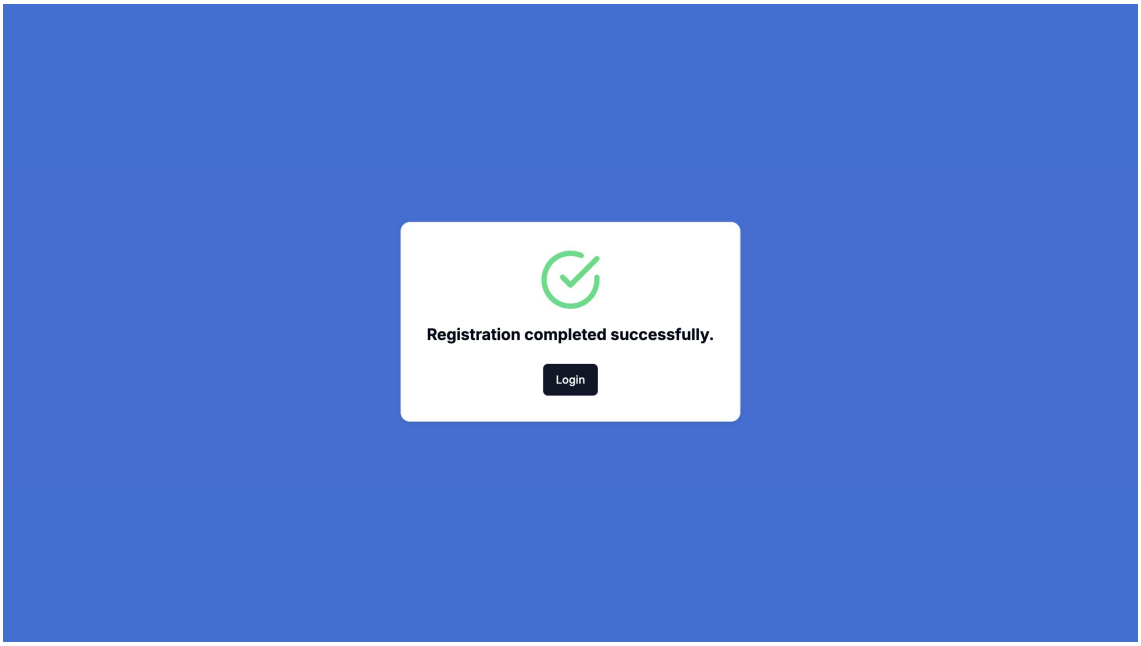
Klikom na "Data Entry" prikazuje se stranica za unos podataka (Slika 7.5), gdje korisnici mogu prijaviti broj kibernetičkih napada koje je njihova tvrtka doživjela u razdoblju od 12 mjeseci. Sučelje je podijeljeno u tablicu koja ima stupce za svaki mjesec u godini, a redovi predstavljaju različite vrste napada. Korisnici unose broj incidenata u odgovarajuća polja, a sustav automatski provjerava valjanost unesenih podataka. U ovom primjeru, vidimo da su unesene negativne vrijednosti za napade "Phishing" u mjesecima svibnju i listopadu, što je odmah označeno crvenim upozorenjem koje poručuje da broj mora biti pozitivan. Također, blokirano je upisivanje slova. Ova validacija osigurava da se unose samo ispravni podaci. Nakon unosa, korisnik može kliknuti gumb "Submit" kako bi poslao podatke. Ukoliko isti korisnik više puta unese podatke, u obzir se uzima samo najnoviji unos te stari unosi ne ulaze u analizu.

Posljednja stranica je "Statistics", koja korisnicima omogućava pregled agregiranih statističkih podataka temeljenih na dostavljenim informacijama o kibernetičkim napadima. Bitno je napomenuti da se statistika prikazuje tek kada postoji minimalno tri unosa podataka od različitih tvrtki, čime se osigurava privatnost podataka prve i druge tvrtke. Slika 7.6 prikazuje slučaj kada ima manje od tri unosa podataka. S druge strane, ukoliko je uvjet od minimalno tri unosa zadovoljen, prikazuju se statistike kao na Slici 7.7. Prikazuju se sljedeći podaci: ukupni broj tvrtki koje su

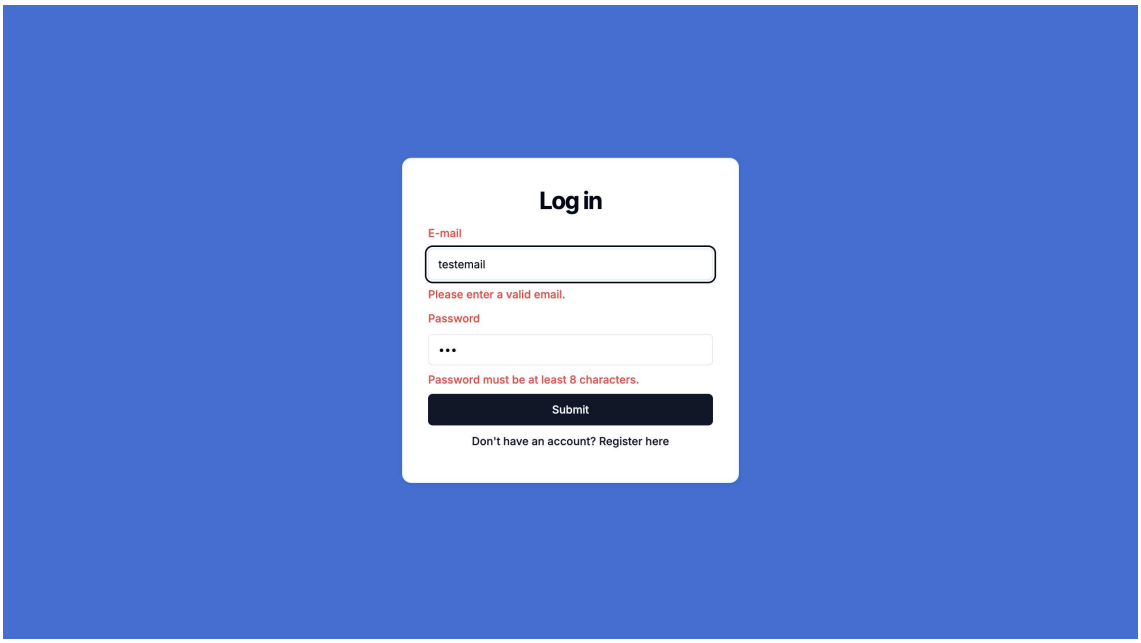
sudjelovale u istraživanju, ukupni broj napada, najčešći napad/i, najrjeđi napad/i i ukupna frekvencija pojavljivanja svakog napada u svakom mjesecu. Kako bi se olakšala preglednost, lebdenjem pokazivača iznad linijskog dijagrama moguće je vidjeti točne frekvencije pojavljivanja svakog od napada.



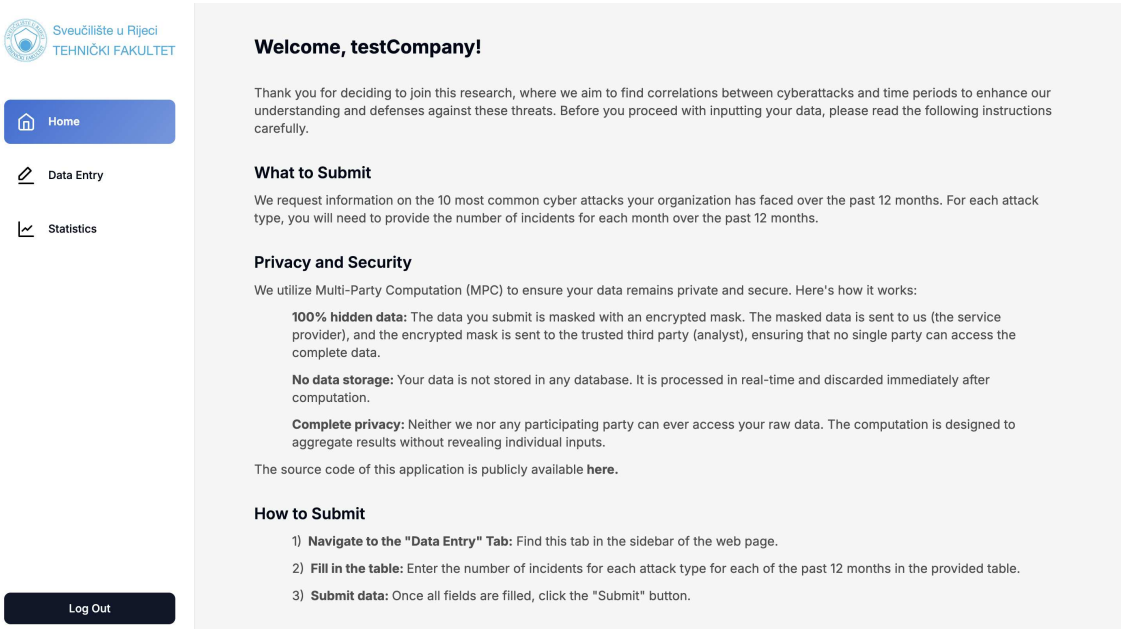
Slika 7.1 Registracija s validacijom.



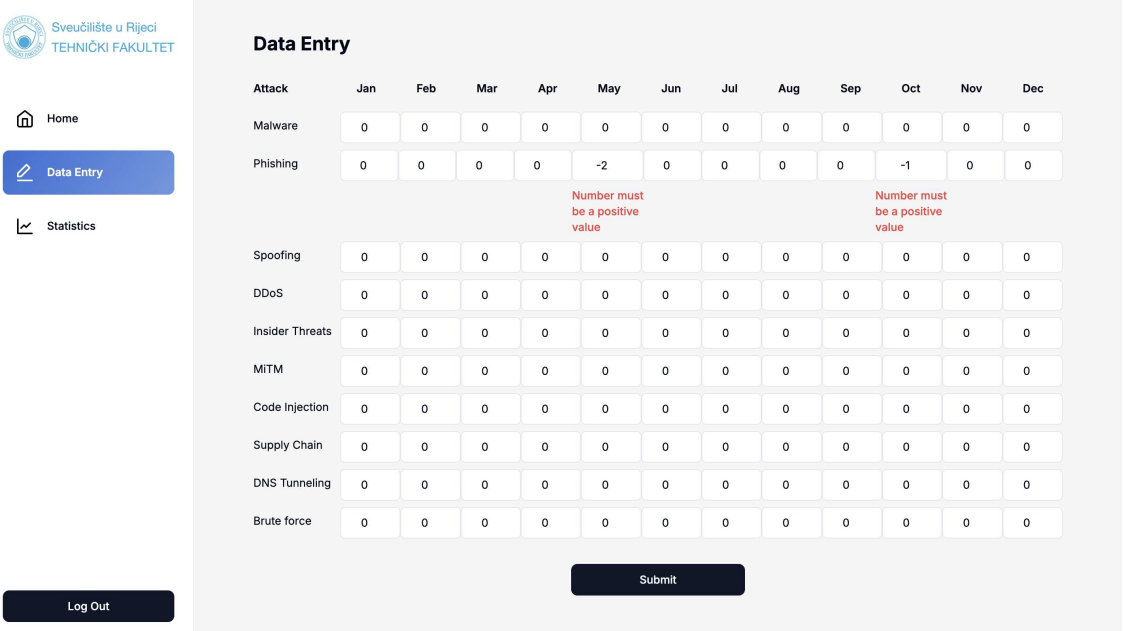
Slika 7.2 Uspješna registracija.



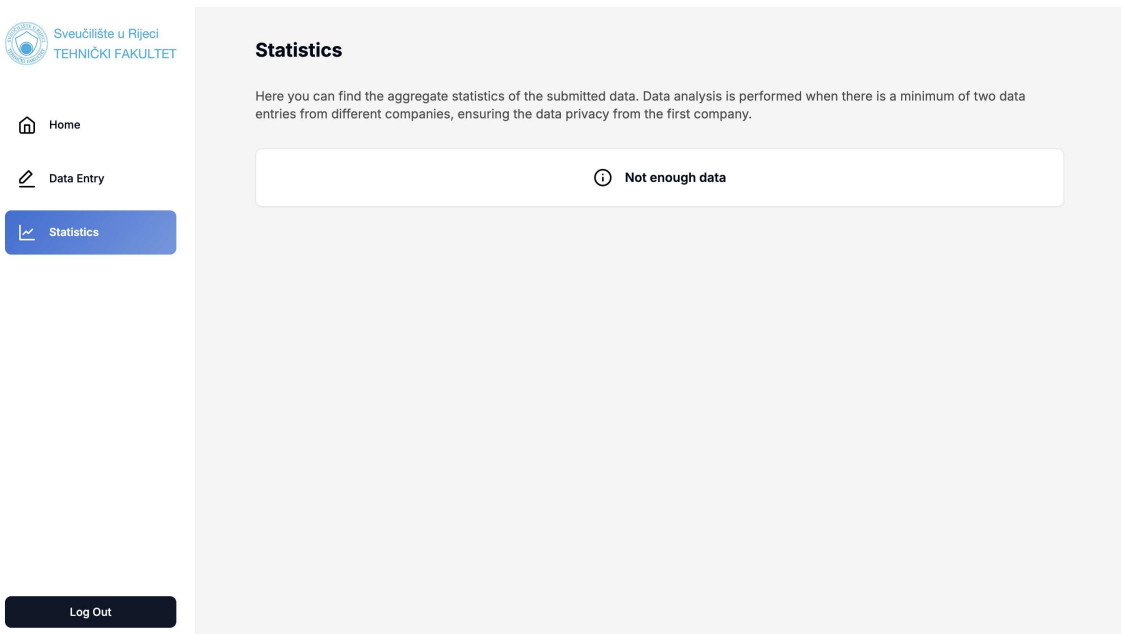
Slika 7.3 Prijava s validacijom.



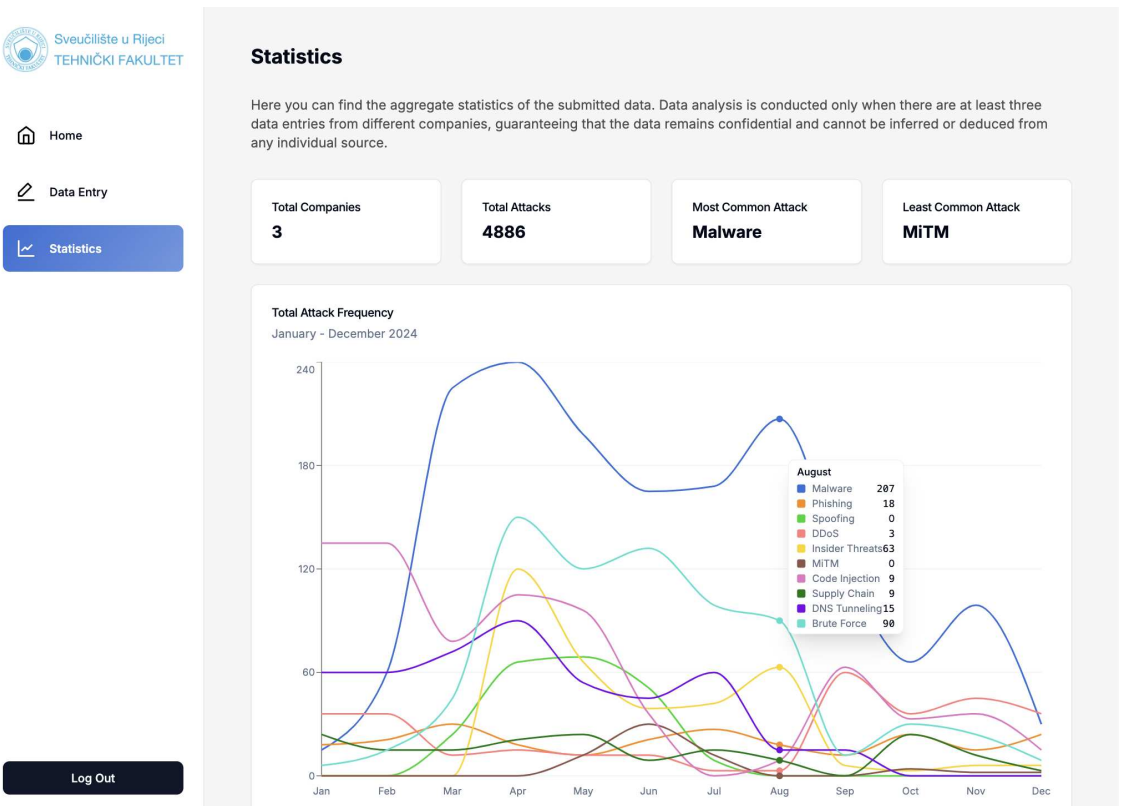
Slika 7.4 Početna stranica.



Slika 7.5 Stranica za unos podataka sa validacijom.



Slika 7.6 Stranica s prikazom statistike kad nema dovoljno podataka.



Slika 7.7 Stranica s prikazom statistike.

Poglavlje 8

Zaključak

Cilj izrađene web platforme bio je omogućiti sigurnu i privatnu obradu podataka o kibernetičkim napadima, čime se osigurava zaštita osjetljivih informacija tvrtki, tj. korisnika. U suvremenom digitalnom okruženju, gdje su podaci neprestano izloženi rizicima, bilo je potrebno razviti sustav koji može prikupljati i analizirati podatke bez ugrožavanja njihove povjerljivosti. Problem je riješen implementacijom MPC tehnologije, koja omogućuje da se statistička obrada podataka provodi na način koji jamči anonimnost i sigurnost. MPC protokoli omogućuju prikupljanje podataka iz različitih izvora, a da pritom niti jedna strana ne može vidjeti podatke drugih sudionika, čime se minimizira rizik od neovlaštenog pristupa i manipulacije. Ovaj pristup također povećava povjerenje korisnika u digitalne platforme koje obrađuju njihove informacije.

Razvoj web platforme koja koristi MPC protokole predstavlja značajan napredak u zaštiti privatnosti tijekom obrade osjetljivih podataka, ali istovremeno donosi i značajne izazove. Troškovi povezani s računalnim resursima potrebnim za kompleksne kriptografske operacije mogu biti visoki, što smanjuje pristupačnost rješenja, osobito za manje organizacije. Uz to, složenost same tehnologije može otežati integraciju u postojeće sustave i zahtijeva visoku razinu tehničkog znanja. Osim tehničkih aspekata, važnost edukacije korisnika ne može se zanemariti. Potrebno je osigurati da krajnji korisnici razumiju kako i zašto se njihovi podaci štite, što zahtijeva jasno predstavljanje funkcionalnosti i prednosti MPC -a. Bez odgovarajuće edukacije i komunikacije, postoji rizik da korisnici neće u potpunosti iskoristiti prednosti koje ova

tehnologija nudi, ili će, zbog nerazumijevanja, izgubiti povjerenje u sustav.

Gledajući unaprijed, ključno je usmjeriti napore prema razvoju sigurnijih i jednostavnijih rješenja koja će omogućiti široku primjenu tehnologija za očuvanje privatnosti u različitim industrijama. Takva rješenja trebaju biti prilagođena ne samo velikim organizacijama s visokim tehničkim kapacitetima, već i manjim tvrtkama koje se suočavaju s ograničenim resursima. Jednostavnost implementacije i korištenja, uz zadržavanje visoke razine sigurnosti, bit će ključni faktori koji će omogućiti širu prihvaćenost ovih tehnologija na globalnoj razini. U konačnici, razvoj ovakvih tehnologija ima potencijal oblikovati digitalni ekosustav u kojem se privatnost i sigurnost podataka ne samo podrazumijevaju, već su i standard, čime se doprinosi globalnoj sigurnosti i održivosti digitalnih inovacija.

Bibliografija

[1] Information Commissioner's Office (2023). Privacy-enhancing technologies (PETs), https://ico.org.uk/media/for-organisations/uk-gdpr-guidance-and-resources/data-sharing/privacy-enhancing-technologies-1-0.pdf, 13. kolovoza 2024.

[2] Matwin, S. (2013). Privacy-Preserving Data Mining Techniques: Survey and Challenges. In: Custers, B., Calders, T., Schermer, B., Zarsky, T. (eds) Discrimination and Privacy in the Information Society. Studies in Applied Philosophy, Epistemology and Rational Ethics, vol 3. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-30487-3_11

[3] Trigyn technologies (2023). Techniques and Challenges for Preserving Privacy in Big Data Analytics, s Interneta, https://www.trigyn.com/insights/techniques-and-challenges-preserving-privacy-big-data-analytics, 13. kolovoza 2024.

[4] European Union Agency for Cybersecurity, ENISA. Readiness Analysis for the Adoption and Evolution of Privacy Enhancing Technologies, s Interneta, https://www.enisa.europa.eu/publications/pets , 13. kolovoza 2024.

[5] OECD (2023). Emerging privacy-enhancing technologies: Current regulatory and policy approaches. OECD Digital Economy Papers, No. 351, OECD Publishing, Paris, https://doi.org/10.1787/bf121be4-en. , 13. kolovoza 2024.

[6] DECENTRIQ. What are privacy-enhancing technologies (PETs)?. , s Interneta, https://www.decentriq.com/article/what-are-privacy-enhancing-technologies, 13. kolovoza 2024.

[7] European Union, Charter of Fundamental Rights of the European Union, 2012/C 326/02, 26 October 2012, s Interneta, https://www.refworld.org/legal/agreements/eu/2012/en/13901, 13. kolovoza 2024.

[8] European Union, Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), 26 October 2012, s Interneta, https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX%3A32016R0679, 13. kolovoza 2024.

[9] Chainlink. (2024). Secure Multi-Party Computation, s Interneta, https://chain.link/education-hub/secure-multiparty-computation-mcp, 22. kolovoza 2024.

[10] Lindell, Y. (2020). Secure multiparty computation (MPC). Cryptology ePrint Archive, Paper 2020/300

[11] Hart, N., & Fatherree, K. (2018, August 24). Secure Multi-Party Computation: A Step Toward a More Secure Digital Future. Bipartisan Policy Center, s Interneta, https://bipartisanpolicy.org/blog/secure-multi-party-computation/?ref=blog.pantherprotocol.io, 22. kolovoza 2024.

[12] Yao, A. C. (1986). How to generate and exchange secrets. Proceedings of the 27th Annual Symposium on Foundations of Computer cience (p./pp. 162–167), Washington, DC, USA: IEEE Computer Society.

[13] Goldreich, O., Micali, S. & Wigderson, A. (1987). How to play ANY mental game or A Completeness Theorem for Protocols ith Honest Majority. Proceedings of the nineteenth annual ACM symposium on Theory of computing (p./pp. 218–229), New York, NY, USA: ACM.

[14] Wikipedia contributors. (2024, August 20). Secure multi-party computation. In Wikipedia, The Free Encyclopedia, s Interneta, https://en.wikipedia.org/w/index.php?title=Secure_multi-party_computation&oldid=1241281555, 22. kolovoza 2024.

[15] Bogetoft, P., Christensen, D. L., Damgård, I., Geisler, M., Jakobsen, T., Krøigaard, M., Nielsen, J. D., Nielsen, J. B., Nielsen, K., Pagter, J., Schwartzbach, M., & Toft, T. (2008). Multiparty computation goes live. Cryptology ePrint Archive, Paper 2008/068.

[16] Qin, L., Lapets, A., Jansen, F., Flockhart, P., Dak Albab, K., Globus-Harris, I., Roberts, S., & Varia, M. (2019, August). From usability to secure computing and back again. In Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019) (pp. 191–210). USENIX Association.

[17] Next.js. The React Framework for the Web, s Interneta, https://nextjs.org/, 15. kolovoza 2024.

[18] Asobo, T. (2024). A Deep dive into CSR, SSR, SSG and ISR, s Interneta, https://dev.to/teyim/a-deep-dive-into-csr-ssr-ssg-and-isr-3513, 15. kolovoza 2024.

[19] Therox, O. (2024). Generics, s Interneta, https://www.typescriptlang.org/docs/handbook/2/generics.html, 15. kolovoza 2024.

[20] Node.js. Introduction to Node.js, s Interneta, https://nodejs.org/en/learn/getting-started/introduction-to-nodejs, 17. kolovoza 2024.

[21] Express.js. Express.js documentation, s Interneta, https://expressjs.com/, 17. kolovoza 2024.

[22] Gillis, A. S. MongoDB, s Interneta, https://www.techtarget.com/searchdatamanagement/definition/MongoDB, 17. kolovoza 2024.

[23] Shadcn UI. (2023). Shadcn UI Documentation, s Interneta, https://ui.shadcn.com/docs, 15. kolovoza 2024.

[24] Emadamerho-Atori, N. (2024). Shadcn UI adoption guide: Overview, examples, and alternatives, s Interneta, https://blog.logrocket.com/shadcn-ui-adoption-guide/, 15. kolovoza 2024.

[25] Anshul, A. (2024). Schema validation in TypeScript with Zod, s Interneta, https://blog.logrocket.com/schema-validation-typescript-zod/, 15. kolovoza 2024.

[26] Wikipedia contributors. (2024, July 2). Bcrypt. In Wikipedia, The Free Encyclopedia. , s Interneta, https://en.wikipedia.org/w/index.php?title=Bcrypt&oldid=1232137191, 15. kolovoza 2024.

[27] Mongoose. Guides, s Interneta, https://mongoosejs.com/docs/guides.html, 16. kolovoza 2024.

[28] Auth0. Node-jsonwebtoken, s Interneta, https://github.com/auth0/node-jsonwebtoken, 18. kolovoza 2024.

Pojmovnik

BSON Binary JSON. 29

CSR Client-Side Rendering. 25

DOM Document Object Model. 25

ENISA The European Union Agency for Cybersecurity. 7

ISR Incremental Static Regeneration. 25

JSON JavaScript Object Notation. 29

JWT JSON Web Token. 22, 23, 28, 36, 51

MPC Secure Multiparty Computation. 1, 2, 11–19, 37, 46, 53, 59

npm Node Package Manager. 27

OECD Organisation for Economic Co-operation and Development. 8

PETs Privacy Enhancing Technologies. 7–10

SSG Static Site Generation. 24, 25

SSR Server-Side Rendering. 24, 25

VSC Visual Studio Code. 49, 50, 52

Sažetak

U ovom diplomskom radu izrađena je web platforma koja koristi Secure Multiparty Computation (MPC) tehnologiju za očuvanje privatnosti. Platforma omogućuje sigurno prikupljanje i analizu podataka o frekvenciji pojavljivanja kibernetičkih napada, osiguravajući da pojedinačni podaci korisnika ostanu zaštićeni. Kroz implementaciju MPC-a, pokazano je kako je moguće sigurno analizirati podatke bez narušavanja povjerljivosti, čime se osigurava visoka razina zaštite privatnosti u digitalnim sustavima.

Ključne riječi **— sigurno višestranačko računanje (MPC), privatnost podataka, agregatna statistika, kibernetički napadi**

Abstract

In this thesis, a web platform was created that uses Secure Multiparty Computation (MPC) technology to preserve privacy. The platform enables the secure collection and analysis of data on the frequency of occurrence of cyberattacks, ensuring that individual user data remains protected. Through the implementation of MPC, it has been demonstrated that it is possible to securely analyze data without breaching confidentiality, thereby ensuring a high level of privacy protection in digital systems.

Keywords **— Secure Multiparty Computation (MPC), data privacy, aggregate statistics, cyber attacks**