**SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEK   
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK**

STRUČNI STUDIJ

A picture containing logo

Description automatically generated

**WEB 3 APLIKACIJA – NADZOR POVIJESTI VOZILA**

SEMINARSKI RAD

Osijek, 2023

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEK  
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH  
TEHNOLOGIJA OSIJEK

STRUČNI STUDIJ

**WEB 3 APLIKACIJA – NADZOR POVIJESTI VOZILA**

SEMINARSKI RAD

Kolegij: PRIMJENA BLOCKCHAIN TEHNOLOGIJE

Mentor: izv. prof. dr. sc. Mirko Köhler

Tim:

1. Srđan Lazić
2. Karlo Buhinjak

Osijek, 2023

**SADRŽAJ**

[UVOD 1](#_Toc137512257)

[1. IZRADA PAMETNOG UGOVORA 2](#_Toc137512258)

[1.1 SOLITITY 2](#_Toc137512259)

[1.2 HARDHAT 5](#_Toc137512260)

[1.3 METAMASK 6](#_Toc137512261)

[1.4 VOLTA TESTNET 8](#_Toc137512262)

[2. IMPLEMENTACIJA UGOVORA U OBLIKU WEB3 APLIKACIJE 10](#_Toc137512263)

[2.1 ARHITEKTURA WEB3 APLIKACIJE 10](#_Toc137512264)

[2.2 FUNKCIJA APLIKACIJE 11](#_Toc137512265)

[2.2.1 OBRAZAC ZA PRIJAVU I REGISTRACIJU 11](#_Toc137512266)

[2.2.2 GLAVNI PROZORAPLIKACIJE – OBRAZAC ZA OBJAVU 15](#_Toc137512267)

[2.2.3 GLAVNI PROZOR APLIKACIJE – LISTA OBJAVA 22](#_Toc137512268)

[2.2.4 GLAVNI PROZOR APLIKACIJE – PRETRAŽIVANJE OBJAVA 25](#_Toc137512269)

[ZAKLJUČAK 27](#_Toc137512270)

[POPIS ILUSTRACIJA 28](#_Toc137512271)

[POPIS PROGRAMSKOG KODA 28](#_Toc137512272)

UVOD

Ovaj seminar je posvećen istraživanju web 3 aplikacija za nadzor povijesti vozila. Bit će obrađeni izazovi koji proizlaze iz manipulacije informacijama o vozilima u online oglasima i kako blockchain tehnologija može riješiti taj problem.

Cilj seminara je pružiti dublje razumijevanje koncepta web 3 aplikacija i njihove uloge u osiguravanju autentičnosti i integriteta informacija o vozilima. Navedeni su konkretni primjeri implementacije i detaljno objašnjeni načini funkcioniranja ove web 3 aplikacije.

Bit će objašnjena arhitektura aplikacije koja je temeljena na node.js s express.js frameworkom. Korišten je Solidity jezik za sastavljanje pametnog ugovora za Volta testnu mrežu. Također, demonstriran je način na koji se koristi MongoDB baza podataka za funkcionalnosti prijave i registracije korisnika.

Posebna pažnja bit će posvećena mogućnosti povezivanja s Metamaskom, koji omogućava korisnicima izvršavanje transakcija na blockchain mreži. Ovaj integrirani sustav pruža sigurnost i transparentnost za transakcije povezane s vozilima, kao što su unos vozila ili provjera vozila.

Tijekom seminara, polaznici će steći dublje razumijevanje koncepta web 3 aplikacija, važnosti blockchain tehnologije te načina na koji se primjenjuje u praktičnom okruženju. Kroz primjere koda i interaktivne demonstracije, polaznici će dobiti uvid u funkcioniranje aplikacije i mogućnosti koje pruža.

Uz raspravu o prednostima i potencijalnim izazovima web 3 aplikacija za nadzor povijesti vozila, ovaj seminar pružit će temelje za daljnje istraživanje i primjenu ove tehnologije u industriji vozila i srodnim područjima.

1. IZRADA PAMETNOG UGOVORA

## 1.1 SOLITITY

Solidity je programski jezik koji se koristi za razvoj pametnih ugovora na Ethereum platformi.

Osnovni cilj Solidity programskog jezika je omogućiti programerima pisanje sigurnih, pouzdanih i efikasnih pametnih ugovora. Solidity kombinira karakteristike jezika poput C++, Pythona i JavaScripta, čineći ga pristupačnim za programere različitih pozadina. Pametni ugovori napisani u Solidity programskom jeziku se kompajliraju u bytecode koji se izvršava na Ethereum Virtual Machine (EVM).

Solidity pruža bogat set alata i biblioteka za razvoj DApps-a (decentralizirane aplikacije) i omogućava programerima da stvaraju inovativne blockchain aplikacije na Ethereum platformi. Kod predstavlja Solidity pametni ugovor za praćenje povijesti automobila na Volta testnoj mreži.

Ugovor sadrži strukturu Car koja opisuje podatke o automobilima kao što su VIN, naziv, marka, motor, kilometraža, vlasnik, godina proizvodnje, pregledao, i datum kreiranja.

Funkcije poput getAllCarsByVin, getAllCarsByReviewedBy i getAllCars omogućavaju dohvaćanje automobila na temelju različitih kriterija.

Funkcija addCar dodaje novi automobil u registar.

Privatne funkcije filterCarsByVin i filterCarsByReviewedBy vrše filtriranje automobila po VIN-u i adminu.

*// SPDX-License-Identifier: GPL-3.0*

**pragma** **solidity** ^0.8.0;

**contract** CarRegistry {

**struct** Car {

**string** vin;

**string** name;

**string** brand;

**string** motor;

**uint256** mileage;

**string** owner;

**uint256** yop;

**string** reviewedBy;

**uint256** createdAt;

    }

    Car[] **public** allCars;

**function** getAllCarsByVin(**string** memory vin) **public** view **returns** (Car[] memory) {

**return** filterCarsByVin(vin);

    }

**function** getAllCarsByReviewedBy(**string** memory reviewedBy) **public** view **returns** (Car[] memory) {

**return** filterCarsByReviewedBy(reviewedBy);

    }

**function** getAllCars() **public** view **returns** (Car[] memory) {

**return** allCars;

    }

**function** addCar(**string** memory vin, **string** memory name, **string** memory brand, **string** memory motor, **uint256** mileage, **string** memory owner, **uint256** yop, **string** memory reviewedBy) **public** {

        Car memory newCar = Car(vin, name, brand, motor, mileage, owner, yop, reviewedBy, block.timestamp);

        allCars.push(newCar);

    }

**function** filterCarsByVin(**string** memory vin) **private** view **returns** (Car[] memory) {

**uint256** count = 0;

**for** (**uint256** i = 0; i < allCars.length; i++) {

**if** (compareStrings(allCars[i].vin, vin)) {

                count++;

            }

        }

        Car[] memory filteredCars = **new** Car[](count);

**uint256** index = 0;

**for** (**uint256** i = 0; i < allCars.length; i++) {

**if** (compareStrings(allCars[i].vin, vin)) {

                filteredCars[index] = allCars[i];

                index++;

            }

        }

**return** filteredCars;

    }

**function** filterCarsByReviewedBy(**string** memory reviewedBy) **private** view **returns** (Car[] memory) {

**uint256** count = 0;

**for** (**uint256** i = 0; i < allCars.length; i++) {

**if** (compareStrings(allCars[i].reviewedBy, reviewedBy)) {

                count++;

            }

        }

        Car[] memory filteredCars = **new** Car[](count);

**uint256** index = 0;

**for** (**uint256** i = 0; i < allCars.length; i++) {

**if** (compareStrings(allCars[i].reviewedBy, reviewedBy)) {

                filteredCars[index] = allCars[i];

                index++;

            }

        }

**return** filteredCars;

    }

**function** compareStrings(**string** memory a, **string** memory b) **private** pure **returns** (**bool**) {

**return** (keccak256(**bytes**(a)) == keccak256(**bytes**(b)));

    }

}

##### Programski kod 1.1 – „Pametni ugovor“

## 1.2 HARDHAT

Hardhat je razvojni okvir (framework) za Ethereum pametne ugovore koji omogućava programerima jednostavno i efikasno razvijanje, testiranje i implementaciju pametnih ugovora. Sa svojim bogatim setom alata i integracija, Hardhat olakšava automatizaciju testiranja, upravljanje okruženjima i interakciju s Ethereum mrežom.

Pomoću naredbe compile (Slika 1.1) kompajlira se pametan ugovor, dok se pomoću naredbe deploy (Slika 1.2) postavlja pametan ugovor na odabranu test mrežu.



#### Slika 1.1 – „compile naredba“



#### Slika 1.2 – „deploy naredba“

## 1.3 METAMASK

Metamask je popularan Ethereum novčanik i ekstenzija preglednika koja omogućava korisnicima jednostavno upravljanje Ethereum novčanicima i interakciju s decentraliziranim aplikacijama (DApps) na Ethereum mreži.

Sa Metamaskom, korisnici mogu sigurno čuvati, slati i primati Ether (ETH) i ERC-20 tokene. Također, Metamask pruža intuitivno sučelje za upravljanje privatnim ključevima, potpisivanje transakcija i pregledanje transakcijske povijesti.

Kroz svoju integraciju s preglednicima, Metamask omogućava korisnicima jednostavan pristup Ethereum ekosustavu i prilagođavanje svojih financijskih aktivnosti na blockchainu.

U ovoj aplikaciji metamask služi za plaćanje transakcija postavljanja zapisa na blockchain.Slika 1.3 prikazuje metamask sučelje, dok slika 1.4 prikazuje upotrebu metamaska u aplikaciji.

A screenshot of a phone

Description automatically generated with medium confidence

#### Slika 1.3 – „Metamask sučelje“

A screenshot of a computer

Description automatically generated with medium confidence

#### Slika 1.4 – „Potvrda transakcije metamask upraviteljom novčanika“

## 1.4 VOLTA TESTNET

Volta testnet je Ethereum testna mreža koja omogućava programerima i korisnicima eksperimentiranje i testiranje aplikacija prije nego što ih implementiraju na glavnoj Ethereum mreži. Volta testnet je kompatibilan s Ethereum Virtual Machine (EVM) i podržava većinu Ethereum alata i pametnih ugovora.

On pruža okruženje za testiranje performansi, funkcionalnosti i interoperabilnosti aplikacija na Ethereumu.

Korištenjem Volta testneta, korisnici mogu provjeriti funkcionalnost svojih pametnih ugovora, provesti testne transakcije i razviti sigurno i pouzdano rješenje prije nego što ga implementiraju na stvarnu Ethereum mrežu.

Volta testnet pretraživaču se može pristupiti putem linka :

<https://volta-explorer.energyweb.org/>

Ugovor je moguće pronaći pod adresom na slici 1.5, a neke od izvedenih transakcija je moguće pogledati u detaljima ugovora kao što je na slici 1.6

A screenshot of a computer

Description automatically generated

#### Slika 1.5 – „Adresa ugovora na testnoj mreži“

A screenshot of a computer

Description automatically generated with medium confidence

#### Slika 1.6 – „Popis transakcija izvedenih na ugovoru“

2. IMPLEMENTACIJA UGOVORA U OBLIKU WEB3 APLIKACIJE

## 2.1 ARHITEKTURA WEB3 APLIKACIJE

Web3 aplikacija omogućava vođenje povijesti automobila, posebno praćenje kilometraže vozila.

Podaci se pouzdano i transparentno pohranjuju na blockchain putem pametnog ugovora koji se nalazi na Volta testnoj mreži.

Za izvršavanje transakcija i zapisivanje podataka koristimo Metamask kao našeg digitalnog novčanika.

Korisnici se mogu registrirati na aplikaciji, a podaci o korisnicima se sigurno pohranjuju u MongoDB bazi podataka. Također, na MongoDB-u se nalaze i administratori koji imaju ovlasti za zapisivanje podataka o automobilima na blockchain.

A diagram of a car management

Description automatically generated with medium confidenceOva kombinacija tehnologija omogućava visoku sigurnost, transparentnost i praktičnost u vođenju povijesti automobila. Arhitektura aplikacije je prikazana na slici 2.1.

#### Slika 2.1 – „Arhitektura aplikacije“

## 2.2 FUNKCIJA APLIKACIJE

### 2.2.1 OBRAZAC ZA PRIJAVU I REGISTRACIJU

Kada korisnik prvi put pristupi aplikaciji, bit će usmjeren na login stranicu. Ovdje će mu biti zatraženo da unese svoje vjerodajnice, kao što su korisničko ime i lozinka. Ovaj korak osigurava da samo ovlašteni korisnici mogu pristupiti aplikaciji i interagirati s njenim funkcionalnostima.

Nakon što korisnik unese svoje vjerodajnice, aplikacija će provjeriti njihovu ispravnost putem baze podataka. Ako su uneseni podaci točni, korisnik će biti usmjeren na glavnu stranicu aplikacije. U suprotnom, bit će obaviješten o pogrešnim podacima i zamoljen da pokuša ponovno.

A screenshot of a login box

Description automatically generated with medium confidence

#### Slika 2.2 – „Stranica prijave u aplikaciju“

Nastavak ovog dijela seminara prikazuje kod koji odgovara "login" stranici web 3 aplikacije. Ovaj kod ima zadatak prikazati formu za unos korisničkih vjerodajnica i omogućiti provjeru ispravnosti unesenih podataka.

Primjer koda za "login" stranicu može izgledati ovako:

<body>

<header>

<div class="container">

<a href="">

<h1>Car Menagment</h1>

</a>

<nav>

<div>

<a href="/api/v1/auth/userlogin">Login</a>

<a href="/api/v1/auth/userregister">Register</a>

</div>

</nav>

</div>

</header>

<form class="login" action="/api/v1/auth/login" method="POST">

<h3>Log In</h3>

<label for="email">Email</label>

<input type="email" name="email" id="" email required />

<label for="password">Password</label>

<input type="password" id="password" name="password" />

<button type="submit">Login</button>

<% if(msg){ %>

<div class="error"><%= msg %></div>

<%} %>

</form>

</body>

##### Programski kod 2.1 – „Obrazac za prijavu“

Prvo se prikazuje "login" predložak koji se sastoji od HTML forme za unos korisničkih vjerodajnica. Nakon što korisnik unese podatke i klikne na gumb "Prijava", POST zahtjev se šalje na rutu koja provjerava ispravnost unesenih podataka.

router.post('/login', async (req, res) => {

const{email, password} = req.body

console.log(req.body)

if (!email || !password) return res.send(400);

const userDB = await User.findOne({ email });

if (!userDB){

//return res.send(401);

req.session.errorMessage = "Incorrect username or password."

res.render("login", {msg: req.session.errorMessage})

}else{

const isValid = comparePassword(password, userDB.password);

if (isValid) {

console.log('Authenticated Successfully!');

req.session.errorMessage = ""

req.session.user = userDB;

res.redirect('/api/v1/post/postList')

} else {

console.log('Failed to Authenticate');

req.session.errorMessage = "Incorrect username or password."

res.render("login", {msg: req.session.errorMessage})

}}

})

##### Programski kod 2.2 – „Zahtjev za prijavu“

Ovaj primjer koda koristi Express.js za upravljanje rutama i prikazivanje predložaka putem render funkcije. Koristi se i body-parser za parsiranje unesenih podataka iz forme.

Koristeći User model, provjerava se postoji li korisnik s unesenim korisničkim imenom u bazi podataka. Ako korisnik ne postoji, prikazuje se poruka o grešci. U suprotnom, koristi se bcrypt biblioteka za usporedbu unesene lozinke s lozinkom pohranjenom u bazi podataka. Ako se lozinke ne podudaraju, također se prikazuje poruka o grešci.

A screenshot of a computer

Description automatically generated with medium confidenceKada korisnik pristupi aplikaciji i nije prijavljen, uvijek mu je pružena opcija za registraciju putem odgovarajuće stranice. Ova stranica će sadržavati formu za unos potrebnih korisničkih podataka kako bi se stvorio novi račun.

#### Slika – 2.3 – „Stranica registracije u aplikaiciju“

router.post('/register', async (req, res) => {

const { email, password} = req.body;

const userDB = await User.findOne({email})

if(userDB) {

//res.status(400).send({msg: 'User alreday exists!'})

res.redirect('/api/v1/auth/userlogin')

}else{

const password = hashPassword(req.body.password)

const isAdmin = false

console.log(password);

const newUser = await User.create({email, password, isAdmin})

newUser.save()

res.redirect('/api/v1/auth/userlogin')

}

})

##### Programski kod 2.3 – „Zahtjev za registraciju“

### 2.2.2 GLAVNI PROZORAPLIKACIJE – OBRAZAC ZA OBJAVU

Kada korisnik pravilno unese svoje vjerodajnice i provjeri se njihova ispravnost, aplikacija će preusmjeriti korisnika na glavnu stranicu koja prikazuje listu objava. Ova stranica može biti implementirana kao API endpoint, što omogućuje korisniku da dobije popis objava u strukturiranom formatu, poput JSON-a.

A screenshot of a computer

Description automatically generated with medium confidence

#### Slika 2.4 – „Glavna stranica aplikacije“

Na glavnoj stranici web 3 aplikacije, korisnicima se pružaju različite funkcionalnosti ovisno o njihovim ulogama. U skladu s tim, aplikacija omogućuje administratorima da na desnoj strani glavne stranice vide formu za unos nove objave. Međutim, ova forma je vidljiva samo kada je korisnik prepoznat kao administrator.

Osim toga, na vrhu glavne stranice nalazi se forma za pretraživanje. Ova forma omogućuje korisnicima da unesu ključne riječi ili kriterije za pretragu objava. Nakon što korisnik unese traženi pojam i klikne na gumb "Pretraži", aplikacija će prikazati rezultate koji zadovoljavaju kriterije pretrage.

U naglasku stranice, korisnici mogu vidjeti listu objava. Ova lista sadrži informacije o svakoj objavi, kao što su naslov, sadržaj ili datum objave. Korisnicima se može pružiti mogućnost pregleda pojedine objave ili izvršavanja drugih akcija na objavama, ovisno o postavkama aplikacije.

<% if (user.isAdmin === true) { %>

<form class="create" id="carForm">

<h3>Add a New Car Record</h3>

<label>Vehicle identification number(VIN):</label>

<input type="text" name="vin" /><br />

<label>Name:</label>

<input type="text" name="name" /><br />

<label>Brand:</label>

<input type="text" name="brand" /><br />

<label>Motor:</label>

<input type="text" name="motor" /><br />

<label>Mileage:</label>

<input type="text" name="mileage" /><br />

<label>Owner:</label>

<input type="text" name="owner" /><br />

<label>Year of production: </label>

<input type="text" name="yop" /><br />

<div class="loading-spinner" id="loadingSpinner"></div>

<button type="submit" id="submitButton">Submit</button>

</form>

<% } %>

##### Programski kod 2.4 – „Obrazac za unos nove objave“

Ovaj programski kod koristi EJS (Embedded JavaScript) kao predložak na frontendu. U prvom retku koristi se EJS sintaksa za uvjetno prikazivanje određenog dijela HTML koda. Ova sintaksa omogućuje nam da dinamički kontroliramo prikaz elemenata ovisno o određenim uvjetima.

Klikom na gumb "Submit" u formi koja je prikazana u kodu, aktivira se skripta koja je postavljena kao listener na događaj slanja obrasca (submit event).

Skripta je podijeljena na tri dijela:

1. Dohvaćanje elementa obrasca, vrtuljka za učitavanje i gumba za slanje:

const carForm = document.getElementById("carForm");

const loadingSpinner = document.getElementById("loadingSpinner");

const submitButton = document.getElementById("submitButton");

##### Programski kod 2.5 – „Dohvaćanje DOM elemenata“

2. Definiranje funkcija za prikazivanje i skrivanje spinnera za učitavanje:

function showLoadingSpinner() {

loadingSpinner.style.display = "block";

}

function hideLoadingSpinner() {

loadingSpinner.style.display = "none";

}

##### Programski kod 2.6 – „Funkcije za prikaz učitavanja“

3. Postavljanje logike prisluškivanja na „potvrdu“ obrasca

carForm.addEventListener("submit", async (event) => {

event.preventDefault();

showLoadingSpinner();

// Show the loading indicator

submitButton.disabled = true;

// Access the form data

const carForm = document.getElementById("carForm");

const formData = new FormData(carForm);

// Retrieve the input values

const vin = formData.get("vin");

const name = formData.get("name");

const brand = formData.get("brand");

const motor = formData.get("motor");

const mileage = formData.get("mileage");

const owner = formData.get("owner");

const yop = formData.get("yop");

const contract\_address = "<%= process.env.CONTRACT\_ADDRESS%>";

const reviewedBy = "<%=user.email%>";

try {

console.log("here");

await ethereum.request({ method: "eth\_requestAccounts" });

const provider = new ethers.providers.Web3Provider(window.ethereum);

const signer = provider.getSigner();

const contractInstance = new ethers.Contract(

contract\_address,

abi,

signer

);

const transaction = await contractInstance.addCar(

vin,

name,

brand,

motor,

mileage,

owner,

yop,

reviewedBy

);

await transaction.wait();

carForm.reset();

} catch (Error) {

console.log(Error);

} finally {

// Hide the loading indicator

hideLoadingSpinner();

submitButton.disabled = false;

window.location.reload();

}

});

##### Programski kod 2.7 – „Objava na blockchain mrežu“

Ova skripta postavlja listener na submit događaj forme. Kada korisnik klikne na gumb "Submit", aktivira se funkcija koja se izvršava asinkrono.

U funkciji, prvo se sprječava preusmjeravanje obrasca (event.preventDefault()) i prikazuje spinner za učitavanje (showLoadingSpinner()). Zatim se dohvaćaju vrijednosti unesene u formu koristeći FormData. Nakon toga slijedi obrada na Ethereum mreži.

Skripta koristi ethereum.request zahtjev za pristup Ethereum računu korisnika. U praksi to znači da će se na ekranu prikazati metamask prozor i od korisnika zatražiti pristupne podatke. Nakon toga, inicijalizira se Web3 provider i signer pomoću ethers.providers.Web3Provider i provider.getSigner(). Slijedi inicijalizacija pametnog ugovora i poziv funkcije addCar na pametnom ugovoru, s proslijeđenim parametrima.

Nakon uspješnog dodavanja automobila, forma se resetira (carForm.reset()), a spinner za učitavanje se skriva (hideLoadingSpinner()) i omogućuje se ponovno aktiviranje gumba za slanje (submitButton.disabled = false). Također, može se obaviti ponovno učitavanje stranice ako je to potrebno (window.location.reload()).

Ovaj kod pokazuje primjer kako se koristi JavaScript i Web3 biblioteka za interakciju s Ethereum mrežom, omogućavajući dodavanje automobila putem pametnog ugovora.

U skriptu koja se izvodi klikom na „potvrdi“, od velike je važnosti priložiti „abi“ pametnog ugovora, koji je potreban prilikom instanciranja, a nalazi se u datoteci zajedno s ugovorom u JSON formatu.

<script src="/abi.js" type="application/javascript"></script>

Osim toga, važno je napomenuti sigurnosne aspekate prilikom rada s Metamaskom i privatnim ključevima. Iz sigurnosnih razloga, nije poželjno pohranjivati privatne ključeve Metamaska u našoj bazi podataka.

Privatni ključ Metamaska predstavlja osnovni mehanizam za pristup i upravljanje korisnikovim računima na Ethereum mreži. Pohranjivanje privatnih ključeva u bazi podataka predstavljalo bi potencijalnu sigurnosnu ranjivost jer bi zlonamjerni napadači mogli pokušati neovlašteno pristupiti tim ključevima.

Umjesto toga, koristi se Metamask kao sučelje za interakciju s privatnim ključevima korisnika. Metamask je razvijen kao sigurna klijentska aplikacija koja drži privatne ključeve na korisnikovom računalu ili mobilnom uređaju. Prilikom izvršavanja transakcija, korisnik će biti upitan za odobrenje i potpisivanje transakcije putem Metamaska, čime se osigurava da privatni ključ ostaje siguran na korisnikovom uređaju.

Tako, sigurnosni aspekti su zaštićeni jer privatni ključevi ostaju u rukama korisnika, a aplikacija koristi Metamask kao sigurno sučelje za interakciju s Ethereum mrežom. To omogućuje pružanje sigurne i pouzdane usluge bez potrebe za pohranjivanjem privatnih ključeva u bazu podataka.

Express.js, kao backend framework, omogućuje obradu zahtjeva i generiranje odgovora na serveru. Međutim, izravna interakcija s Metamaskom putem backenda nije moguća jer Metamask je klijentska aplikacija koja radi na korisnikovom pregledniku.

Da bi se postigla željena funkcionalnost, potrebno je uspostaviti međuprostor između backenda i frontenda koji će omogućiti komunikaciju s Metamaskom. Na frontendu upotrebom JavaScript i Web3 biblioteka može se ostvarili interakcija s Ethereum mrežom putem Metamaska.

Kada korisnik pristupi aplikaciji i želi izvršiti određenu radnju koja zahtijeva interakciju s Metamaskom, frontend će inicirati zahtjev za dohvaćanjem podataka s Metamaska, poput adrese korisnika i potpisivanja transakcija. Zatim će podaci biti poslani na Ethereum mrežu radi izvršavanja željene operacije.

Ovaj međuprostor omogućuje interakciju između klijentskog dijela aplikacije, Metamaska i Ethereum mreže. Express.js, kao backend framework, i dalje ima svoju važnost u obradi drugih zahtjeva i pružanju potrebne logike na serveru, ali za interakciju s Metamaskom i Ethereum mrežom, koristi se JavaScript na frontendu.

Ova arhitektura omogućuje da se integrira Metamask i Ethereum funkcionalnost u aplikaciju, pružajući korisnicima mogućnost interakcije s blockchain mrežom i potvrdu transakcija na siguran način.

### 2.2.3 GLAVNI PROZOR APLIKACIJE – LISTA OBJAVA

Na glavnoj stranici aplikacije, osim forme za unos nove objave na blockchain mrežu, također se nalazi lista postojećih objava koje su dohvaćene s mreže. Ova lista objava pruža korisnicima pregled prethodno objavljenih informacija o vozilima.

Kada se lista objava dohvati s mreže, prikazuje se na glavnoj stranici našeg web sučelja. Korisnici mogu pregledavati informacije o vozilima koje su objavljene na blockchain mreži, poput identifikacijskog broja vozila (VIN), naziva, marke, motora, prijeđene kilometraže, vlasnika i godine proizvodnje.

Ova lista objava omogućuje korisnicima da prate povijest vozila koja su objavljena na blockchain mreži. Također, korisnicima pruža transparentnost i pouzdanost informacija o vozilima koje su dostupne za pregled.

<% if (cars.length !== 0) { %> <% for (let i = cars.length - 1; i >=0;

i--) { %>

<div class="workout-details">

<h4><%= cars[i].vin %></h4>

<p><strong>Name: </strong><%= cars[i].name %></p>

<p><strong>Brand: </strong><%= cars[i].brand %></p>

<p><strong>Motor: </strong><%= cars[i].motor %></p>

<p><strong>Mileage: </strong><%= cars[i].mileage %></p>

<p><strong>Owner: </strong><%= cars[i].owner %></p>

<p><strong>Year of production: </strong><%= cars[i].yop %></p>

<p><strong>Reviewed By: </strong><%= cars[i].reviewedBy %></p>

<p>

<strong>Review date: </strong>

<%= new Date(cars[i].createdAt \* 1000).toLocaleString() %>

</p>

</div>

<% } %>

##### Programski kod 2.8 – „Kartica za prikaz informacija vozila“

const provider = new ethers.providers.JsonRpcProvider(process.env.API\_URL);

const contractInstance = new ethers.Contract(process.env.CONTRACT\_ADDRESS, abi, provider);

router.get('/postList', async(req, res) => {

const user = req.session.user

if (user.isAdmin) {

try {

var cars = await contractInstance.getAllCarsByReviewedBy(user.email)

res.render('index', {cars, user, searchParameter:null})

} catch (error){

console.error(error);

res.status(500).json({ message: "Failed to fetch cars" });

}

} else {

try {

var cars = await contractInstance.getAllCars();

res.render('index', {cars, user, searchParameter:null})

} catch (error){

console.error(error);

res.status(500).json({ message: "Failed to fetch cars" });

}

}

})

##### Programski kod 2.9 – „Upit na dohvaćanje podataka s mreže“

U ovom dijelu koristi se ethers.js biblioteku za uspostavljanje veze s Ethereum mrežom i komunikaciju s pametnim ugovorom.

Prvo se stvara instanca JsonRpcProvider objekta koji pruža pristup Ethereum mreži putem JSON-RPC protokola. Koristi se vrijednost process.env.API\_URL koja predstavlja URL adresu Ethereum čvora (node) koji se koristi za komunikaciju s mrežom. U ovom slučaju to je volta testna mreža.

Zatim se stvara instanca pametnog ugovora (Contract) koristeći Contract razred ethers.js biblioteke. Prilikom stvaranja instance pametnog ugovora, prosljeđuju se tri parametra: process.env.CONTRACT\_ADDRESS, koji sadrži adresu na kojoj je pametni ugovor implementiran na Ethereum mreži, abi koji predstavlja sučelje pametnog ugovora, i provider koji pruža vezu s Ethereum mrežom.

Nadalje, u rutini router.get('/postList'), koja se aktivira prilikom zahtjeva za prikazivanje liste objava na glavnoj stranici, provjerava se ulogu korisnika. Ako je korisnik administrator (user.isAdmin je istinito), pametni ugovor dohvaća objave koje su pregledane od strane tog korisnika (odgovarajući e-mail adresi), koristeći metodu getAllCarsByReviewedBy. U suprotnom slučaju, metoda getAllCars dohvaća sve objave bez obzira na korisnika koji ih je pregledao.

Nakon dohvata objava, metoda res.render prikazuje index.ejs predložak (template) s podacima o objavama (cars), trenutno prijavljenom korisniku (user) i parametru za pretraživanje (searchParameter). Ova metoda generira HTML prikaz koji se šalje kao odgovor na zahtjev korisnika.

Važno je napomenuti da za prikazivanje liste objava s mreže nije potrebno dohvaćati novčanik ili izvršavati potvrdu transakcija. Dohvaćanje podataka s blockchaina, u ovom slučaju objava vozila, smatra se besplatnom operacijom koja ne zahtijeva plaćanje naknade ili potvrdu transakcije. Time korisnici mogu besplatno pregledavati informacije o vozilima koja su pohranjena na Ethereum mreži putem aplikacije.

### 2.2.4 GLAVNI PROZOR APLIKACIJE – PRETRAŽIVANJE OBJAVA

Forma za pretraživanje je smještena na vrhu stranice kako bi bila lako dostupna i vidljiva korisnicima. Sadrži polje za unos pretraživačkog pojma, specifično broja vozila (VIN). Nakon što korisnik unese pretraživački pojam i pritisne tipku za pretraživanje, aplikacija šalje zahtjev serveru kako bi dohvatila objave koje odgovaraju unesenom kriteriju.

Ova funkcionalnost pretraživanja omogućuje korisnicima da brzo pronađu specifične objave koje ih zanimaju, olakšavajući im pregled i pronalaženje željenih informacija o vozilima. Bez potrebe za pregledom cijele liste objava, korisnici mogu koristiti ovu formu za fokusirano pretraživanje i dohvaćanje relevantnih rezultata.

<form

class="search-form"

action="/api/v1/post/postSearch"

method="POST"

>

<input type="text" placeholder="Search by VIN" name="search" />

<button type="submit"><i class="fa fa-search"></i></button>

</form>

##### Programski kod 2.9 – „Obrazac za pretraživanje“

router.post('/postSearch', async(req, res) => {

const user = req.session.user

const searchParameter = req.body.search

try {

var cars = await contractInstance.getAllCarsByVin(searchParameter);

res.render('index', { cars, user, searchParameter });

} catch (error){

console.error(error);

res.status(500).json({ message: "Failed to fetch cars" });

}

})

##### Programski kod 3.0 – „Upit na pretraživanje objava“

Ova ruta se aktivira kada korisnik pošalje zahtjev s pretraživačkim pojmom putem HTTP POST metode.

Kada korisnik pošalje zahtjev, server prima informaciju o pretraživačkom pojmu iz tijela zahtjeva (req.body.search). Pretraživački pojam predstavlja unos korisnika i sadržava informacije o broju vozila (VIN). Nakon što se dobije pretraživački pojam, aplikacija koristi contractInstance.getAllCarsByVin() metodu kako bi dohvatila sve objave koje odgovaraju tom kriteriju. Ova metoda koristi pametni ugovor (smart contract) i blockchain mrežu za dohvat podataka.

Ako se dohvat objava uspješno izvrši, aplikacija koristi metodu res.render() kako bi prikazala ažuriranu stranicu 'index'. U kontekstu ovog prikaza, predaju se objekti cars, user i searchParameter kako bi se ažurirala lista objava, podaci o trenutno prijavljenom korisniku i pretraživački pojam koji je korišten.

U slučaju bilo kakve pogreške tijekom dohvata objava, aplikacija obrađuje grešku i šalje odgovarajući status i poruku o grešci korisniku.

ZAKLJUČAK

U zaključku ovog seminara o web 3 aplikacijama za nadzor povijesti vozila, bit će sažeti ključni elemente koji su istraženi i prezentirani.

Bitno je razumjeti izazove povezane s manipulacijom informacija o vozilima u online oglasima i prepoznajemo važnost osiguravanja integriteta tih informacija. Kroz primjenu blockchain tehnologije, predstavljena je inovativna web 3 aplikaciju koja pruža rješenje za ovaj problem.

Već postoje slična komercijalna rješenja koja se naplaćuju kako bi se osigurala autentičnost informacija o vozilima. Međutim, za potrebe seminara i iz vlastite želje za istraživanjem, implementirano je vlastito rješenje koje demonstrira primjenu web 3 tehnologija u praksi.

Predstavljena web 3 aplikacija temelji se na node.js platformi s express.js frameworkom. Kroz korištenje Solidity jezika, stvoren je pametni ugovor na volta testnoj mreži, dok su podaci korisnika pohranjeni i upravljani putem MongoDB baze podataka. Integriranjem Metamaska omogućeno korisnicima izvršavanje transakcija na blockchain mreži, pružajući sigurnost i transparentnost.

Seminar je pružio detaljan uvid u koncept web 3 aplikacija, blockchain tehnologiju i njihovu primjenu u nadzoru povijesti vozila. Kroz prezentaciju primjera koda i objašnjenja arhitekture aplikacije, polaznici su stekli razumijevanje o načinu funkcioniranja aplikacije i koristima koje pruža.

Važno je istaknuti da web 3 tehnologije imaju širu primjenu u raznim industrijama i pružaju transparentnost, autentičnost i sigurnost podataka. Ova aplikacija predstavlja samo jedno od mogućih rješenja u svijetu nadzora povijesti vozila. Postoji mnogo drugih komercijalnih rješenja koja su već dostupna na tržištu.

Ovaj seminar je bio prilika za istraživanje načina na koji web 3 tehnologije mogu riješiti probleme integriteta informacija.

POPIS ILUSTRACIJA

[Slika 1.1 – „compile naredba“ 5](#_Toc137511990)

[Slika 1.2 – „deploy naredba“ 5](#_Toc137511991)

[Slika 1.3 – „Metamask sučelje“ 7](#_Toc137511992)

[Slika 1.4 – „Potvrda transakcije metamask upraviteljom novčanika“ 7](#_Toc137511993)

[Slika 1.5 – „Adresa ugovora na testnoj mreži“ 9](#_Toc137511994)

[Slika 1.6 – „Popis transakcija izvedenih na ugovoru“ 9](#_Toc137511995)

[Slika 2.1 – „Arhitektura aplikacije“ 10](#_Toc137511996)

[Slika 2.2 – „Stranica prijave u aplikaciju“ 11](#_Toc137511997)

[Slika – 2.3 – „Stranica registracije u aplikaiciju“ 14](#_Toc137511998)

[Slika 2.4 – „Glavna stranica aplikacije“ 15](#_Toc137511999)

POPIS PROGRAMSKOG KODA

[Programski kod 1.1 – „Pametni ugovor“ 4](#_Toc137512245)

[Programski kod 2.1 – „Obrazac za prijavu“ 12](#_Toc137512246)

[Programski kod 2.2 – „Zahtjev za prijavu“ 13](#_Toc137512247)

[Programski kod 2.3 – „Zahtjev za registraciju“ 14](#_Toc137512248)

[Programski kod 2.4 – „Obrazac za unos nove objave“ 16](#_Toc137512249)

[Programski kod 2.5 – „Dohvaćanje DOM elemenata“ 17](#_Toc137512250)

[Programski kod 2.6 – „Funkcije za prikaz učitavanja“ 17](#_Toc137512251)

[Programski kod 2.7 – „Objava na blockchain mrežu“ 18](#_Toc137512252)

[Programski kod 2.8 – „Kartica za prikaz informacija vozila“ 22](#_Toc137512253)

[Programski kod 2.9 – „Upit na dohvaćanje podataka s mreže“ 23](#_Toc137512254)

[Programski kod 2.9 – „Obrazac za pretraživanje“ 25](#_Toc137512255)

[Programski kod 3.0 – „Upit na pretraživanje objava“ 26](#_Toc137512256)