VIŠJA STROKOVNA ŠOLA ACADEMIA

MARIBOR

RAZVOJ DECENTRALIZIRANE FINANCE (DEFI) SPLETNE APLIKACIJE

Kandidat: Vasil Trajkovski

Vrsta študija: študent izrednega študija

Študijski program: Računalništvo

Mentor predavatelj: mag. Dušan Brglez

Mentor v podjetju: mag. Ervin Schaff

Lektorica: dr. Petra Kunc, prof. slov.

Maribor, 2026

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Podpisani Vasil Trajkovski sem avtor diplomskega dela z naslovom Razvoj decentralizirane finance (DeFi) spletne aplikacije, ki sem ga napisal pod mentorstvom mag. Dušana Brgleza.

S svojim podpisom zagotavljam, da:

- je predloženo delo izključno rezultat mojega dela
- sem poskrbel, da so dela in mnenja drugih avtorjev, ki jih uporabljam v predloženi nalogi, navedena oz. citirana skladno s pravili Višje strokovne šole Academia Maribor
- se zavedam, da je plagiatorstvo – predstavljanje tujih del oz. misli, kot moje lastne kaznivo po Zakonu o avtorski in sorodnih pravicah (Uradni list RS, št. 16/07 – uradno prečiščeno besedilo, 68/08, 110/13, 56/15 in 63/16 – ZKUASP); prekršek pa podleže tudi ukrepom Višje strokovne šole Academia Maribor skladno z njenimi pravili
- skladno z 32.a členom ZASP dovoljujem Višji strokovni šoli Academia Maribor objavo diplomskega dela na spletnem portalu šole.

Maribor, junij 2026 Podpis študenta:

ZAHVALA

Iskreno se zahvaljujem mentorju mag. Dušanu Brglezu za strokovno vodenje, koristne napotke in podporo pri nastajanju tega diplomskega dela.

Zahvala gre tudi somentorju mag. Ervinu Schaffu za praktične izkušnje in usmeritve pri razvoju aplikacije.

Posebna zahvala pripada moji družini, ki mi je v času študija stala ob strani in me spodbujala pri doseganju zastavljenih ciljev. Zahvaljujem se tudi prijateljem in sošolcem za spodbudne besede in pomoč v zahtevnejših trenutkih.

Razvoj decentralizirane finančne spletne aplikacije je predstavljal tehničen in vsebinski izziv, ki ne bi bil mogoč brez podpore vseh navedenih.

POVZETEK

Diplomsko delo obravnava razvoj decentralizirane finance (DeFi) spletne aplikacije, katere namen je prikazati povezavo med teoretičnimi izhodišči decentraliziranih financ in praktično implementacijo delujoče prototipne rešitve. V teoretičnem delu so predstavljeni osnovni koncepti blockchain tehnologije, pametnih pogodb, decentraliziranih financ, ključnih funkcionalnosti DeFi aplikacij ter varnostnih tveganj, ki spremljajo razvoj takšnih sistemov. Poseben poudarek je namenjen vlogi omrežja Ethereum, standardu ERC-20, avtomatiziranim ustvarjalcem trga (AMM), *staking* mehanizmom, *yield farming* pristopu in pomenu varnosti pri razvoju decentraliziranih aplikacij.

V praktičnem delu je bila razvita prototipna DeFi spletna aplikacija, ki temelji na omrežju Ethereum in vključuje več funkcionalnih modulov. Jedro sistema predstavljajo pametne pogodbe, razvite v programskem jeziku Solidity. Implementirane so bile žetonska pogodba ERC-20, pogodba za likvidnostni bazen z DEX logiko, *staking* pogodba in *yield farming* pogodba. Uporabniški vmesnik je bil razvit z uporabo ogrodja React in programskega jezika TypeScript, pri čemer je bila posebna pozornost namenjena modularnosti, preglednosti transakcij in varni uporabniški interakciji. Za povezavo med *frontendom*, kripto denarnico in blockchain omrežjem je bila vzpostavljena Web3 integracija, hkrati pa je bil vključen tudi podporni *backend* sloj, namenjen agregaciji podatkov in pripravi analitičnih prikazov uporabnika.

Pomemben del diplomskega dela predstavlja tudi varnostni vidik razvoja. Pri implementaciji so bili uporabljeni osnovni varnostni mehanizmi, kot so validacija podatkov, zaščita pred *reentrancy* napadi, ločevanje *approval* in *action* transakcij, uporaba preverjenih knjižnic ter preverjanje pravilnega omrežja. Na ta način je bilo mogoče zmanjšati nekatera ključna tveganja, ki so značilna za DeFi okolje. Aplikacija je bila testirana v lokalnem razvojnem okolju z uporabo orodja Hardhat, izvedeni pa so bili *unit* testi pametnih pogodb, *frontend* preverjanje, uporabniško testiranje in osnovna varnostna analiza.

Rezultati diplomskega dela kažejo, da je mogoče tudi v okviru prototipne rešitve uspešno povezati več ključnih DeFi funkcionalnosti v enoten sistem.

Ključne besede: *decentralizirane finance, Ethereum, pametne pogodbe, staking, yield farming*

ABSTRACT

Development of a Decentralized Finance (DeFi) Web Application

This thesis addresses the development of a decentralized finance (DeFi) web application, with the main purpose of connecting the theoretical foundations of decentralized finance with the practical implementation of a functional prototype solution. The theoretical part presents the core concepts of blockchain technology, smart contracts, decentralized finance, key DeFi application functionalities, and the security risks associated with the development of such systems. Special emphasis is placed on the role of the Ethereum network, the ERC-20 standard, automated market makers (AMM), staking mechanisms, yield farming, and the importance of security in the development of decentralized applications.

The practical part includes the development of a prototype DeFi web application based on the Ethereum network, consisting of several functional modules. The core of the system is represented by smart contracts developed in the Solidity programming language. The implemented contracts include an ERC-20 token contract, a liquidity pool contract with DEX logic, a staking contract, and a yield farming contract. The user interface was developed using React and TypeScript, with particular attention given to modularity, transaction transparency, and secure user interaction. A Web3 integration layer was implemented to connect the frontend, the wallet, and the blockchain network, while a supporting backend layer was also included for data aggregation and analytical data presentation.

An important part of the thesis is the security aspect of development. During implementation, several fundamental security mechanisms were applied, including input validation, reentrancy protection, the separation of approval and action transactions, the use of verified libraries, and network verification. These measures helped reduce some of the major risks typical of the DeFi environment. The application was tested in a local development environment using Hardhat, and the testing process included smart contract unit tests, frontend verification, user testing, and basic security analysis.

The results of the thesis show that even within the scope of a prototype solution, it is possible to successfully combine multiple key DeFi functionalities into a single system.

Keywords: *decentralized finance, Ethereum, smart contracts, staking, yield farming*

KAZALO VSEBINE

1 UVOD 15

1.1 Opis področja in opredelitev problema 15
1.2 Namen, cilji in osnovne trditve 15
1.3 Predpostavke in omejitve 16
1.4 Uporabljene raziskovalne metode 17

2 TEORETIČNA PODLAGA 18

2.1 Tehnologija blockchain in pametne pogodbe 18
2.2 Decentralizirane finance (DeFi) – koncept, razvoj in pomen 19
2.3 Ključne funkcionalnosti DeFi spletnih aplikacij 22
2.3.1 Decentralizirana izmenjava (DEX) 22
2.3.2 Posojanje in izposoja sredstev 24
2.3.3 Staking in yield farming 25
2.3.4 DAO – decentralizirana avtonomna organizacija 28
2.4 Tehnološki sklad spletnih DeFi aplikacij (Solidity, Web3, Javascript, Node.js, ipd.) 30
2.4.1 Veriga blokov (Blockchain) plast in pametne pogodbe 31
2.4.2 Integracijska plast (Ether.js, denarnice, RPC) 31
2.4.3 Uporabniški vmesniki (Frontend – React + TypeScript) 32
2.4.4 Backend kot podporni sloj 32
2.4.5 Testno okolje in razvojna orodja 33

3 KIBERNETSKA VARNOST V DEFI SPLETNIH APLIKACIJAH 34

3.1 Tipične grožnje DeFi spletne aplikacije 34
3.2 Varnost pametnih pogodb (ranljivosti, revizije, orodja) 36
3.3 Varnost spletnega vmesnika (frontend) in zaščita uporabnika 37
3.4 Varnost komunikacije med frontend - smart contracts - backend 38
3.5 Pripomočki in prakse za zagotavljanje kibernetske varnosti 40
3.6 Primer vdorov v DeFi spletne aplikacije in analize posledic 41

4 NAČRTOVANJE SPLETNE APLIKACIJE 42

4.1 Definicija funkcionalnih zahtev za spletno DeFi rešitev 42
4.2 Zasnova uporabniškega vmesnika in uporabniške izkušnje (UI/UX) 44
4.3 Načrt arhitektur spletne aplikacije 46
4.3.1 Frontend (React/HTML/CSS + Web3.js/ Ether.js) 47
4.3.2 Backend (Node.js/Express, povezava z blockchain) 48
4.3.3 Blockchain del (pametna pogodba) 49

4.4 Poslovna logika in uporabniški primeri uporabe 50
4.4.1 Uporabniški primer: Povezava z denarnico 50
4.4.2 Uporabniški primer: Zamenjava žetonov (DEX swap) 51
4.4.3 Uporabniški primer: Dodajanje likvidnosti 51
4.4.4 Uporabniški primer: Staking 52
4.4.5 Uporabniški primer: Yield farming 52
4.4.6 Uporabniški primer: Dvig sredstev 53
5 RAZVOJ SPLETNE DEFI APLIKACIJE 54
5.1 Razvoj pametnih pogodb v jeziku Solidity 54
5.1.1 Žetonska pogodba 55
5.1.2 Pogodba za likvidnostni bazen 56
5.1.3 Pogodba za staking 58
5.1.4 Pogodba za yield farming 60
5.1.5 Varnostni mehanizmi pri implementaciji pametnih pogodb 61
5.2 Izdelava spletnega uporabniškega vmesnika (frontend) 64
5.3 Razvoj povezave frontend – blockchain (Web3 integracija) 67
5.4 Backend logika in upravljanje podatkov 71
5.5 Testno okolje: simulacija blockchaina (Hardhat/Ganache) 73
5.6 Povezovanje vseh komponent v funkcionalno celoto 76
6 TESTIRANJE IN ANALIZA DELOVANJA 79
6.1 Testiranje pametnih pogodb (unit test, edge case testiranje) 79
6.1.1 Uporabljen testni pristop in ogrodje 79
6.1.2 Testirane funkcionalnosti po posameznih pogodbah 81
6.1.3 Najpomembnejši edge case scenariji 82
6.1.4 Rezultati unit testiranja 83
6.2 Testiranje frontend funkcionalnosti 85
6.3 Uporabniško testiranje 88
6.4 Varnostno testiranje (audit, analiza napak) 92
6.5 Analiza rezultatov in primerjava z obstoječimi rešitvami 93
6.6 Odgovori na raziskovalna vprašanja in preverjanje hipotez 95
6.7 Prednosti, slabosti in možnosti izboljšav spletne aplikacije 96
6.8 Predlogi za nadaljnji razvoj 97
7 SKLEP 99
8 VIRI IN LITERATURA 101
9 PRILOGE 105

KAZALO SLIK

SLIKA 1: ARHITEKTURA DEFI 22
SLIKA 2: IMPLEMENTACIJA ŽETONSKE POGODBE ERC20TOKEN NA OSNOVI STANDARDA ERC-20 56
SLIKA 3: IZSEK KODE PAMETNE POGODBE ZA LIKVIDNOSTNI BAZEN Z LOGIKO ZA DODAJANJE LIKVIDNOSTI 57
SLIKA 4: NADALJEVANJE IZSEKA KODE PAMETNE POGODBE ZA LIKVIDNOSTNI BAZEN 58
SLIKA 5: IZSEK KODE PAMETNE POGODBE ZA STAKING 59
SLIKA 6: IZSEK KODE PAMETNE POGODBE ZA STAKING Z LOGIKO ZA CLAIM NAGRAD 60
SLIKA 7: IZSEK KODE PAMETNE POGODBE ZA YIELD FARMING 61
SLIKA 8: MODULARNA ZASNOVA PAMETNIH POGODB IN NJIHOVE FUNKCIONALNE POVEZAVE 63
SLIKA 9: POVEZAVA FUNKCIONALNOSTI APLIKACIJE IN PAMETNIH POGODB 64
SLIKA 10: ZAČETNI ZASLON APLIKACIJE Z NADZORNO PLOŠČO IN PREGLEDOM PORTFELJA 65
SLIKA 11: DEX MODUL ZA ZAMENJAVO ŽETONOV 66
SLIKA 12: MODUL ZA DODAJANJE IN ODSTRANJEVANJE LIKVIDNOSTI 66
SLIKA 13: MODULA ZA STAKING IN YIELD FARMING 67
SLIKA 14: OPOZORILO PRI VISOKI TOLERANCI ZDRSA CENE 67
SLIKA 15: DIAGRAM POTEKA, IZVEDBA ZAMENJAVE ŽETONOV 69
SLIKA 16: DIAGRAM POTEKA, STAKING TRANSAKCIJA 70
SLIKA 17: STRUKTURA BACKEND PROJEKTA 72
SLIKA 18: ARHITEKTURNI DIAGRAM CELOTNE APLIKACIJE 73
SLIKA 19: PRVI DEL DEPLOY SKRIPTE ZA LOKALNO TESTNO OKOLJE 75
SLIKA 20: DRUGI DEL DEPLOY SKRIPTE Z INICIALIZACIJO PROTOKOLA IN IZPISOM NASLOVOV POGODB 76
SLIKA 21: UPORABNIŠKI PRIMERI UPORABE RAZVITE DEFI APLIKACIJE 77
SLIKA 22: DEFINIRAN UKAZ ZA POGANJANJE UNIT TESTOV V OKOLJU HARDHAT 80
SLIKA 23: KONFIGURACIJA HARDHAT RAZVOJNEGA OKOLJA IN TESTNEGA OKOLJA 80
SLIKA 24: ORGANIZACIJA UNIT TESTOV PO POSAMEZNIH POGODBAH 81
SLIKA 25: PRIMER EDGE CASE TESTA ZA PREVERJANJE SLIPPAGE ZAŠČITE PRI ZAMENJAVI SREDSTEV 82
SLIKA 26: PRIMER EDGE CASE TESTA ZA OMEJITEV IZPLAČILA NAGRAD NA FINANCIRANI REWARD POOL 83
SLIKA 27: REZULTAT ZA USPEŠNO IZVEDENE TESTE 1 85
SLIKA 28: REZULTAT ZA USPEŠNO IZVEDENE TESTE 2 85
SLIKA 29: PRIMER AVTOMATIZIRANEGA TESTIRANJA UPORABNIŠKEGA VMESNIKA V REACT OKOLJU 88

KAZALO TABEL

TABELA 1: VARNOSTNI MEHANIZMI 62
TABELA 2: PREGLED UNIT TESTOV PO PAMETNIH POGODBAH 83
TABELA 3: POGLED TESTIRANJA FRONTEND FUNKCIONALNOSTI 86
TABELA 4: PREGLED SCENARIJEV UPORABNIŠKEGA TESTIRANJA 89
TABELA 5: KLJUČNI VARNOSTNI MEHANIZMI 93
TABELA 6: PRIMERJAVA RAZVITE APLIKACIJE Z IZBRANIMI REFERENČNIMI DEFI REŠITVAMI 94

KAZALO GRAFIKONOV

GRAFIKON 1: REZULTAT UNIT TESTOV PO POGODBAH 84
GRAFIKON 2: STRUKTURA TESTOV PO POGODBAH GLEDE NA TIP SCENARIJA 84
GRAFIKON 3: REZULTATI AVTOMATIZIRANEGA FRONTEND TESTIRANJA PO MODULIH 87
GRAFIKON 4: POVPREČEN ČAS IZVEDBE POSAMEZNIH UPORABNIŠKIH NALOG 91
GRAFIKON 5: ŠTEVILO NAPAK PO POSAMEZNIH UPORABNIŠKIH NALOGAH 91

Uporabljeni angleški izrazi in kratice:

- AMM: Automated Market Maker (avtomatizirani ustvarjalci trga)
- API: uporabljeno v kontekstu vmesnikov zalednega sistema za integracijo in zaščito
- Approval: odobritev oziroma dovoljenje pametni pogodbi za upravljanje sredstev
- APR/APY: Annual Percentage Rate / Annual Percentage Yield (letni odstotek donosa)
- Backend: zaledni sistem (podporni sloj spletne aplikacije)
- Bitcoin (BTC): kriptovaluta in izvorno sredstvo na verigi blokov Bitcoin
- Blockchain: tehnologija veriženja blokov
- CEX: Centralized Exchange (centralizirana borza)
- ChainId: identifikator omrežja
- Check-effect-interaction in Mutex: varnostna vzorca pri razvoju pametnih pogodb za preprečevanje napadov
- CI/CD: postopki avtomatizacije in uvedbe aplikacije za večjo zanesljivost
- Clone strani: zlonamerne kopije legitimnih strani, ustvarjene z namenom zavajanja uporabnikov
- DAO: Decentralized Autonomous Organization (decentralizirana avtonomna organizacija)
- DeFi: Decentralized Finance (decentralizirane finance)
- DEX: Decentralized Exchange (decentralizirana borza)
- Edge cases: robni primeri, ki se preverjajo med testiranjem logike pametnih pogodb
- ERC-20: standard za žetone na omrežju Ethereum
- Ethereum (ETH): blockchain omrežje in izvorna kriptovaluta (ETH)
- EVM: Ethereum Virtual Machine (virtualni stroj, ki deluje na Ethereum tehnologiji)
- Fiat: tradicionalne valute (npr. ameriški dolar)
- Flash loan: hitra posojila brez zavarovanja znotraj iste transakcije
- Frontend/Front-end: uporabniški vmesnik spletne aplikacije

- Gas/Gas fee: stroški izvedbe transakcije na omrežju
- Hard fork: razkol v skupnosti in tehnološka nadgradnja omrežja
- Impermanent loss: začasna izguba vrednosti sredstev v likvidnostnem skladu v primerjavi s preprostim držanjem žetonov
- JavaScript / Node.js / React / TypeScript / Solidity: programski jeziki in ogrodja tehnološkega sklada
- KYC: Know Your Customer (preverjanje identitete)
- LP/LP Tokens: Liquidity provider (ponudniki likvidnosti) in likvidnostni žetoni
- LTV: Loan-to-Value (razmerje med posojilom in vrednostjo zavarovanja)
- Man-in-the-middle: vrsta napada pri komunikaciji prek neustrezno konfiguriranih orodij
- OpenZeppelin / Ether.js / Web3.js / Hardhat / Slither / Mythril: razvojna in varnostna orodja ter knjižnice
- Order book: knjiga naročil
- P2P: Peer-to-Peer omrežje
- Phishing: vrsta napada z zavajanjem za pridobitev sredstev ali ključev
- PoS: Proof-of-Stake (metoda soglasja oziroma dokazilo o deležu)
- Price oracles: oraklji, ki pametnim pogodbam zagotavljajo zunanje podatke o cenah sredstev.
- Reentrancy: vrsta varnostne ranljivosti in napada s ponovnim vstopom v pametno pogodbo
- RPC: Remote Procedure Call (mehanizem za komunikacijo z vozlišči)
- Rug pull: vrsta zlonamerne sheme oziroma kraje sredstev s strani razvijalcev
- Slippage/slippage tolerance: zdrs cene oziroma razlika med pričakovano in dejansko izvršeno ceno ter njena toleranca
- Smart contracts: pametne pogodbe
- SPA: Single Page Application (enoslojna spletna aplikacija)

- Staking: vlaganje oziroma zaklepanje žetonov za zagotavljanje varnosti ali pridobivanje nagrad
- Supply-chain: dobavna veriga znotraj razvoja programske opreme
- Swap: zamenjava žetonov
- Token/tokenomika: digitalni žeton in z njim povezana ekonomska strategija
- TVL: Total Value Locked (skupna zaklenjena vrednost v sistemu)
- UI/UX: User Interface (uporabniški vmesnik) in User Experience (uporabniška izkušnja)
- XSS: Cross-Site Scripting (vrsta napada na uporabniški vmesnik)
- Yield farming: donosno kmetovanje oziroma maksimiranje donosov z zagotavljanjem likvidnosti
- Analytics: analitika oziroma pregled obdelanih podatkov
- Allowance: dovoljeni znesek žetonov, ki ga lahko pametna pogodba porabi v imenu uporabnika
- Audit: varnostni pregled kode, pametnih pogodb ali sistema
- Balance: stanje sredstev na naslovu ali v pogodbi
- Bridge: mehanizem za prenos sredstev ali podatkov med različnimi blockchain omrežji
- Burn/burning: uničenje žetonov oziroma trajno zmanjšanje njihove količine v obtoku
- Claim/claim rewards: zahtevek za izplačilo nagrad iz *staking* ali *yield farming* pogodbe
- Connector/wallet connector: povezovalni mehanizem med aplikacijo in kripto denarnico
- Data aggregation: združevanje in priprava podatkov iz več virov v enoten prikaz
- Deploy/deployment: postopek namestitve pametne pogodbe ali aplikacije v izbrano okolje
- Event/blockchain event: dogodek, ki ga pametna pogodba zabeleži ob izvedbi določene funkcije
- Express: spletno ogrodje za Node.js, namenjeno izdelavi strežniških aplikacij in API-jev
- Fallback RPC: rezervni RPC-ponudnik oziroma nadomestna povezava ob izpadu primarnega ponudnika

- Farming rewards: nagrade, ki jih uporabnik prejema za sodelovanje v *yield farming* mehanizmu
- Funding reward pool: polnjenje nagradnega sklada za izplačevanje nagrad
- Hook: programska funkcija oziroma mehanizem v React okolju za upravljanje stanja, učinkov ali logike
- Injected wallet: denarnica, ki je v brskalnik dodana prek razširitve, na primer MetaMask
- Liquidity provider: uporabnik, ki v likvidnostni bazen zagotavlja sredstva
- Liquidity pool reserves: rezerve sredstev, shranjene v likvidnostnem bazenu
- nonReentrant: varnostni mehanizem oziroma oznaka funkcije, ki preprečuje ponovni vstop v funkcijo med izvajanjem
- Mainnet: glavno produkcijsko blockchain omrežje
- Portfolio: pregled uporabnikovih sredstev, vložkov in nagrad
- Position/staking position/farming position: uporabniška pozicija v *staking* ali *yield farming* modulu
- Reward pool: sklad sredstev, iz katerega se izplačujejo nagrade uporabnikom
- Reward token: žeton, ki se uporablja za izplačevanje nagrad
- Routing: iskanje poti menjave med več likvidnostnimi pari z namenom boljše izvedbe transakcije
- SafeERC20: varnostna knjižnica za varnejšo uporabo ERC-20 funkcij
- Sepolia: Ethereum testno omrežje za razvoj in preverjanje aplikacij
- Smoke test/smoke testing: osnovno preverjanje, ali ključne funkcionalnosti sistema delujejo
- State/smart contract state: trenutno stanje podatkov, shranjenih v pametni pogodbi ali aplikaciji
- TransferFrom: ERC-20 funkcija, ki omogoča prenos žetonov na podlagi predhodne odobritve
- Unit test: Enotski test za preverjanje pravilnosti posamezne funkcije ali modula

- Wagmi: Knjižnica za React aplikacije, namenjena povezavi z denarnicami in Web3 interakciji
- Viem: Knjižnica za Ethereum komunikacijo, branje podatkov in pošiljanje transakcij

1 UVOD

1.1 Opis področja in opredelitev problema

Decentralizirane finance (DeFi) predstavljajo rastoče področje finančnih tehnologij, ki temelji na tehnologiji verige blokov (blockchain) in pametnih pogodbah za izvajanje finančnih storitev brez centralnih posrednikov. Za razliko od tradicionalnih sistemov, kjer je zaupanje vezano na centralne institucije, DeFi temelji na kriptografsko preverljivih pravilih in odprtokodni kodi.

Kljub potencialu pa DeFi aplikacije prinašajo izzive na področju varnosti pametnih pogodb, ranljivosti vmesnikov in kompleksnosti integracije tehnoloških slojev. Številne platforme so bile že tarče napadov, kar kaže, da decentralizacija sama po sebi ne zagotavlja varnosti.

To diplomsko delo obravnava DeFi spletne aplikacije, ki združujejo decentralizirano izmenjavo, *staking* in *yield farming*, pri čemer je poseben poudarek na tehnični zanesljivosti, varnosti in uporabniški izkušnji. Cilj ni zgolj opisati teoretičnih konceptov, temveč razviti konkretno spletno aplikacijo na omrežju Ethereum in analizirati ključne arhitekturne ter varnostne odločitve.

1.2 Namen, cilji in osnovne trditve

Namen diplomskega dela je razviti funkcionalno in varno spletno aplikacijo s področja decentraliziranih financ (DeFi), ki vključuje ključne funkcionalnosti, kot so decentralizirana izmenjava (DEX), *staking* in *yield farming*. Poleg implementacije je cilj naloge analizirati arhitekturne in varnostne odločitve, ki vplivajo na stabilnost in zanesljivost takšnih sistemov.

Cilj naloge je pridobiti poglobljeno razumevanje tehnologij, ki omogočajo delovanje DeFi aplikacij, zlasti blockchain infrastrukture, pametnih pogodb v jeziku Solidity ter integracije z uporabniškim vmesnikom. Poseben poudarek je namenjen varnosti, saj so ranljivosti pametnih pogodb in napake v integraciji med najpogostejšimi vzroki za izgubo sredstev.

V okviru naloge se preverjajo naslednje trditve: vključitev ključnih DeFi funkcionalnosti povečuje uporabno vrednost aplikacije; pravilna integracija pametnih pogodb z uporabniškim vmesnikom zmanjšuje napake in izboljšuje skalabilnost (prilagajanje povečani obremenitvi), preglednost transakcij vpliva na zaupanje uporabnikov ter implementacija varnostnih praks pomembno zmanjšuje tveganje napadov.

Hipoteze, ki jih bom skozi potek diplomskega dela zavrgel ali potrdil, so:

- H1: Vključitev ključnih DeFi funkcionalnosti (DEX, *staking*, posojanje) povečuje vrednost aplikacije.
- H2: Pravilna integracija pametnih pogodb z UI/*backend* sistemom zmanjša število napak ali dovoli skalabilno delovanje.
- H3: Uporabniška izkušnja in preglednost transakcij povečata zaupanje in angažiranost uporabnikov.
- H4: Dodatna hipoteza – BTEC 2025 »Kibernetska varnost«: Implementacija kibernetskih varnostnih praks (*smart contract audit*, CI/CD zaščita, *frontend* varnost) znatno zmanjša tveganje napadov v DeFi aplikaciji.

Raziskovalna vprašanja, ki jih bom ovrednotil:

- Kako vključitev ključnih funkcionalnosti decentraliziranih financ, kot so decentralizirana izmenjava (DEX), *staking* in posojanje, vpliva na povečanje uporabne vrednosti in celovitosti spletne aplikacije?
- Na kakšen način pravilna arhitekturna integracija pametnih pogodb z uporabniškim vmesnikom in zalednim sistemom prispeva k zmanjšanju števila programskih napak ter omogoča skalabilno delovanje sistema?
- Kako uporabniška izkušnja in transparentnost transakcij vplivata na povečevanje zaupanja ter angažiranosti uporabnikov znotraj decentraliziranega okolja?
- V kolikšni meri dosledna implementacija kibernetskih varnostnih praks, kot so zaščita uporabniškega vmesnika, pregledi pametnih pogodb in CI/CD zaščita, zmanjšuje tveganje za uspešne napade na DeFi aplikacijo?
- Kateri so ključni arhitekturni in tehnološki izzivi pri razvoju in implementaciji prototipne DeFi spletne aplikacije na omrežju Ethereum?

1.3 Predpostavke in omejitve

Pri izdelavi diplomskega dela se izhaja iz predpostavke, da blockchain infrastruktura omrežja Ethereum zagotavlja zanesljivo okolje za izvajanje pametnih pogodb ter da varnost sistema primarno izhaja iz pravilne zasnove in implementacije kode. Predpostavlja se tudi, da

uporabniki razpolagajo z osnovnim znanjem uporabe kripto denarnic in razumevanjem delovanja decentraliziranih aplikacij.

Razvita aplikacija predstavlja prototipno rešitev in ne produkcijskega finančnega sistema. V okviru naloge ne bo izvedena zunanja varnostna revizija, temveč bodo uporabljena razvojna in analitična orodja za preverjanje varnosti pametnih pogodb. Implementacija bo potekala na testnem omrežju, zato aplikacija ne bo upravljala realnih finančnih sredstev.

1.4 Uporabljene raziskovalne metode

Diplomsko delo temelji na analizi strokovne in znanstvene literature s področja blockchain tehnologije, decentraliziranih financ in varnosti pametnih pogodb. Teoretična izhodišča so bila uporabljena kot podlaga za razvoj prototipne DeFi aplikacije na omrežju Ethereum. V praktičnem delu je bila uporabljena metoda razvoja programske rešitve, ki vključuje implementacijo pametnih pogodb, integracijo z uporabniškim vmesnikom ter testiranje funkcionalnosti in varnosti sistema.

2 TEORETIČNA PODLAGA

2.1 Tehnologija blockchain in pametne pogodbe

Tehnologija veriženja blokov (angl. *Blockchain*) predstavlja temelj decentraliziranih aplikacij in decentraliziranih financ (DeFi). Gre za porazdeljeno zbirko podatkov, ki jo sestavljajo zaporedno povezani bloki transakcij. Vsak blok vsebuje časovni žig, podatke o transakcijah ter kriptografsko zgoščeno vrednost (*hash*) prejšnjega bloka, kar zagotavlja nespremenljivost zapisov (Nakamoto, b. d., str. 2; Ethereum Whitepaper, 2026). Zaradi takšne strukture je naknadna sprememba podatkov praktično nemogoča brez soglasja omrežja.

Blockchain omrežja običajno delujejo v *peer-to-peer* (P2P) okolju, kjer vsa vozlišča sledijo protokolu soglasja za potrjevanje novih blokov. Ta decentralizirana arhitektura omogoča visoko stopnjo odpornosti proti manipulacijam ter bizantinsko toleranco napak. Sistem lahko deluje pravilno tudi v primeru, ko nekatera vozlišča posredujejo napačne ali zlonamerne podatke. (Nakamoto, b. d., str. 3; Ethereum Whitepaper, 2026).

Pomemben prelom v razvoju tehnologije je predstavljal pojav kriptovalute Bitcoin leta 2008, ki je prvi uspešno rešil problem dvojne porabe brez centralnega nadzora. Nadaljnji razvoj pa je omogočil Ethereum, ki je blockchain nadgradil z možnostjo izvajanja pametnih pogodb (Nakamoto, b. d., str. 1; Ethereum Whitepaper, 2026).

Pametne pogodbe so programski protokoli, ki se izvajajo na blockchainu in samodejno izvršujejo vnaprej določena pravila. Njihovo delovanje je deterministično, kar pomeni, da bo pri enakem vhodu rezultat vedno enak, kar zagotavlja predvidljivost in zanesljivost sistema (Schär, 2021, str. 1–2). Izvajajo se v okolju Ethereum Virtual Machine (EVM), kjer vsako vozlišče omrežja preverja pravilnost izvajanja (Ethereum, 2026; Ethereum Whitepaper, 2026).

V nasprotju s klasičnimi spletnimi aplikacijami, kjer uporabnik mora zaupati ponudniku storitve, so pametne pogodbe javno dostopne in preverljive. Koda je zapisana na blockchainu, zato je njeno delovanje transparentno in nadzorovano s pravili soglasja omrežja (Schär, 2021, str. 1–2).

S tem pametne pogodbe prevzemajo vlogo tradicionalnih finančnih institucij, vendar delujejo na podlagi odprtokodne kode in kriptografske varnosti.

Kljub prednostim pa pametne pogodbe predstavljajo tudi varnostna tveganja. Ker so po objavi na blockchainu praviloma nespremenljive, lahko programske napake povzročijo resne finančne izgube. Kot kažejo raziskave, so ranljivosti, kot so napadi ponovnega vstopa (*reentrancy attacks*), nepravilno upravljanje dostopnih pravic ter neustrezna obravnava zunanjih klicev, med najpogostejšimi vzroki za varnostne incidente v ekosistemu decentraliziranih financ (DeFi) (Marchesi idr., 2020, str. 5).

Zato je pri razvoju decentralizirane finančne aplikacije ključnega pomena pravilna zasnova pametnih pogodb, uporaba preverjenih knjižnic ter dosledno testiranje. V okviru tega diplomskega dela bodo pametne pogodbe razvite v jeziku Solidity, implementirane na omrežju Ethereum in predstavljene kot temelj poslovne logike razvite DeFi spletne aplikacije.

2.2 Decentralizirane finance (DeFi) – koncept, razvoj in pomen

Pomen

Decentralizirane finance (DeFi) predstavljajo ekosistem finančnih aplikacij (finančno infrastrukturo), zgrajenih na veriženju blokov (blockchain), ki omogoča izvajanje finančnih storitev brez centralnih posrednikov. Temeljijo na pametnih pogodbah, ki avtomatizirajo procese, kot so trgovanje, posojanje, upravljanje likvidnosti in distribucija nagrad (Schär, 2021, str. 1–20). Najpomembnejša platforma za razvoj DeFi rešitev je Ethereum, ki zaradi podpore pametnim pogodbam omogoča izvajanje kompleksnih finančnih mehanizmov.

Ena izmed ključnih prednosti DeFi je, da vzpostavi odprt, dostopen in brezmejni finančni sistem brez posrednikov, kjer lahko vsak uporabnik z internetno povezavo sodeluje brez preverjanja identitete ali centralnega nadzora (Schär, 2021, str. 8). Tak pristop povečuje dostopnost, transparentnost in nadzor uporabnika nad lastnimi sredstvi.

Ključne komponente DeFi ekosistema (TRM Labs, 2025; Stellar Development Foundation, 2025):

- ***Decentralizirane borze (DEX):*** so platforme, ki uporabnikom zagotavljajo večjo avtonomijo in zmanjšujejo tveganje nasprotne stranke, saj omogočajo trgovanje s kriptovalutami med posamezniki brez uporabe običajnih centraliziranih posrednikov.
- ***Platforme za posojanje in izposojanje:*** napredne storitve DeFi, ki uporabnikom brez uporabe običajnih finančnih posrednikov omogočajo izposojo sredstev ob predložitvi zavarovanja ali posojanje svojih kriptovalutnih sredstev za ustvarjanje obresti prek pametnih pogodb.

- ***Stabilni kovanci (angl. stablecoins):*** razred kriptovalut, zasnovanih tako, da ohranjajo stabilno vrednost, pogosto temeljijo na fiat valutah, kot je ameriški dolar in ponujajo prednosti tehnologije veriženja blokov, hkrati pa služijo kot povezava med nestanovitnimi trgi kriptovalut ter tradicionalnimi financami.
- ***Donosno kmetovanje (angl. yield farming):*** v ekosistemu DeFi se *yield farming* nanaša na napredne naložbene tehnike, pri katerih uporabniki maksimirajo donosnost kriptovalutnih sredstev z zagotavljanjem likvidnosti več protokolom v zameno za nagrade, pogosto v obliki transakcijskih provizij ali žetonov za upravljanje.
- ***Sintetična sredstva:*** vrednost drugih sredstev, kot so kriptovalute in tradicionalnimi finančnimi inštrumenti, je predstavljena s tokeniziranimi derivati, ki zagotavljajo izpostavljenost različnim trgom, ne da bi zahtevali neposredno lastništvo osnovnih sredstev (TRM Labs, 2025).

Decentralizirane finance v primerjavi s tradicionalnimi financami zagotavljajo številne prednosti, kot so (TRM Labs, 2025):

- ***Dostopnost:*** ne glede na geografsko lokacijo ali finančno situacijo so storitve DeFi na voljo vsem z internetno povezavo.
- ***Preglednost:*** z omogočanjem dostopa do vseh transakcij in kode pametnih pogodb v blockchainu, preglednost spodbuja odgovornost in zaupanje.
- ***Interoperabilnost:*** kompleksno finančno blago in storitve je mogoče ustvariti s protokoli DeFi, ki med seboj enostavno komunicirajo.
- ***Inovacije:*** Odprtokodna narava DeFi spodbuja hitre inovacije in eksperimentiranje v finančni industriji.

Razvoj

Razvoj DeFi je povezan z globalno finančno krizo leta 2008, ki je razkrila ranljivosti tradicionalnega finančnega sistema. Blockchain tehnologija je ponudila alternativo, kjer se zaupanje prenaša iz centralnih institucij na kriptografske protokole (OSL, 2025).

Pomemben mejnik je bil razvoj avtomatiziranih ustvarjalcev trga (AMM)**,** ki so omogočili decentralizirano trgovanje brez klasične knjige naročil. AMM uporabljajo matematične formule za določanje cen v likvidnostnih skladih, kar je omogočilo razmah decentraliziranih borz (OSL, 2025).

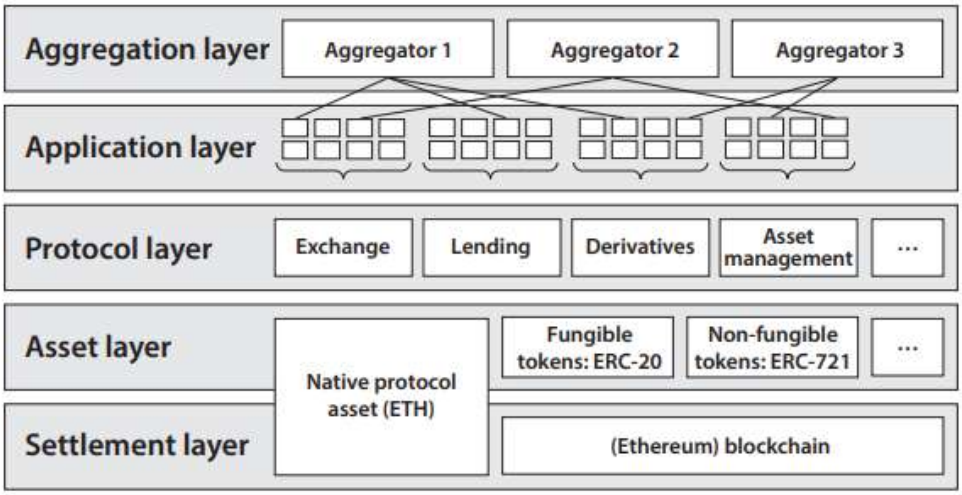
Pozneje se je razvil tudi koncept *yield farming*, ki uporabnikom omogoča optimizacijo donosov z zagotavljanjem likvidnosti različnim protokolom. Čeprav je ta model povečal kapitalske

tokove v DeFi, je hkrati prinesel večja varnostna tveganja in kompleksnost upravljanja (OSL, 2025).

Koncept

Arhitektura DeFi je večplastna in temelji na hierarhični strukturi, ki jo podrobno opisuje Schär (2021), in sicer razlikujemo pet plasti:

- **Plast poravnave (1. plast)** sestavljajo Bitcoin (BTC) na blockchainu Bitcoin in Ethereum (ETH) na blockchainu Ethereum, ki sta primera sredstev izvornega protokola. Plast poravnave omrežju omogoča varno vzdrževanje podatkov o lastništvu in zagotavlja, da vse spremembe njegovega stanja sledijo njegovim pravilom. Blockchain lahko štejemo za temelj izvajanja brez zaupanja, ki deluje kot plast za poravnavo in reševanje sporov.
- **Plast sredstev (2. plast)** sestavljajo vsa sredstva, ki so izdana na vrhu poravnalne plasti. Ta zajema sredstva izvornega pretoka kot vsa druga sredstva (žetoni), ki so izdana v blockchainu.
- **Plast protokola (3. plast)** ponuja smernice za posebne primere uporabe, kot so upravljanje premoženja v verigi, izvedeni finančni instrumenti, dolžniški trgi in decentralizirane borze. Za izvajanje teh standardov se običajno uporablja zbirka pametnih pogodb, do katerih lahko dostopa vsak uporabnik ali tudi aplikacija DeFi. Posledično so ti protokoli zelo interoperabilni.
- **Aplikacijska plast (4. plast)** ustvarja uporabniško usmerjene programe, ki se povezujejo z določenimi protokoli. Spletni vmesnik, ki temelji na brskalniku, običajno abstrahira interakcijo pametnih pogodb in poenostavi protokole.
- **Agregacijska plast (5. plast)**. Agregatorji gradijo platforme, osredotočene na uporabnika, ki se povezujejo z različnimi protokoli in aplikacijami. S hkratnim povezovanjem z več protokoli uporabnikom omogočajo opravljanje sicer zahtevnih dejavnosti, ponujajo orodja za primerjavo in ocenjevanje storitev ter strnjujejo ustrezne podatke na razumljiv način.



Slika 1: Arhitektura DeFi

(Vir: https://berkeley-defi.github.io/assets/material/Fabian-Schar-decentralized-finance-on-blockchain-and-smart-contract-based-financial-markets.pdf, 2026)

2.3 Ključne funkcionalnosti DeFi spletnih aplikacij

Decentralizirane finance temeljijo na naboru funkcionalnosti, ki jih omogočajo pametne pogodbe na blockchain omrežju. Med najpomembnejše sodijo decentralizirana izmenjava (DEX), posojanje in izposoja sredstev, *staking* mehanizmi, *yield farming* ter decentralizirano upravljanje prek DAO struktur. Te funkcionalnosti predstavljajo osnovo večine sodobnih DeFi protokolov in so hkrati temelj razvijane aplikacije v okviru tega diplomskega dela (Schär, 2021, str. 5–19).

2.3.1 Decentralizirana izmenjava (DEX)

Decentralizirana borza (DEX) omogoča neposredno trgovanje med uporabniki brez centralnega posrednika. Namesto centralizirane knjige naročil (*order book*), DEX pogosto uporablja avtomatizirane ustvarjalce trga (AMM), ki določajo ceno sredstev na podlagi matematične formule in razmerja v likvidnostnem skladu (Schär, 2021, str. 1–10).

Pri modelu AMM uporabniki zagotavljajo likvidnost v obliki dveh žetonov, v zameno pa prejmejo likvidnostne žetone (*LP tokens*), ki predstavljajo njihov delež v skladu. Vsaka menjava generira provizijo, ki se razdeli med ponudnike likvidnosti (Schär, 2021, str. 13–15).

Pomemben element, ki zaznamuje delovanje DEX, je pojav *slippage* (zdrs cene), ki nastane, ko se cena med potrditvijo transakcije spremeni zaradi volatilnosti ali omejene likvidnosti. Uporabnik nastavi toleranco zdrsa, kar predstavlja zaščitni mehanizem pred neugodnimi cenovnimi spremembami (Schär, 2021, str. 14–16).

Za razvito aplikacijo to pomeni implementacijo:

- pametne pogodbe za upravljanje likvidnostnega sklada;
- mehanizma za izračun razmerij med sredstvi;
- varnostnih preverjanj pred izvršitvijo izmenjave.

Prednosti DEX

Borze DEX postajajo vse bolj priljubljene kot uspešen način interakcije s kriptovalutami, saj potrošnikom zagotavljajo številne pomembne prednosti (CryptoCloud, 2025):

- **Popoln nadzor nad denarjem:** Čeprav so uradno v lasti borze, se sredstva na centraliziranih borzah hranijo v uporabniških računih. Uporabnik lahko v primeru kibernetskega napada izgubi denar. Lastnik pa ohrani popoln nadzor nad borzami kriptovalut DEX.
- **Izboljšana zaščita:** DEX so manj dovzetni za kibernetske napade, ker se ne zanašajo na centralizirane strežnike. Decentralizirano borzo lahko ogrozi le ranljivost pametne pogodbe, ne pa strojna oprema strežnika.
- **Transakcijska nespremenljivost in preglednost:** Ker je vsaka transakcija shranjena v blockchainu, je ni mogoče spremeniti ali odstraniti. Manipulacija s cenami in obsegom trgovanja ni mogoča.
- **Odsotnost posrednikov:** Vsak uporabnik lahko trguje, ker DEX ne dovoljuje bankam ali drugim centraliziranim institucijam sodelovanja v borzni dejavnosti.
- **Zasebnost in anonimnost:** Ker večina borz DEX ne zahteva preverjanja KYC, so privlačne za potrošnike, ki spoštujejo zasebnost.

Slabosti DEX

Kljub prednostim, ki jih imajo decentralizirane borze, so tukaj tudi pomanjkljivosti, ki lahko predstavljajo izzive za določene skupine uporabnikov (CryptoCloud, 2025):

- **Začetniki težko uporabljajo borzo:** Razumevanje konceptov veriženja blokov, delovanja pametnih pogodb in konfiguriranje denarnic s kriptovalutami so predpogoji za trgovanje na DEX.
- **Omejena likvidnost**: V primerjavi s centraliziranimi sistemi imajo nekatere borze DEX manjši obseg trgovanja. Ko velike transakcije pomembno vplivajo na menjalni čas, to povzroči zdrs cen.
- **Fiat valute niso podprte**: Uporabniki morajo kupiti sredstva kriptovalut na CEX prek platform P2P, ker DEX poslujejo samo s kriptovalutami.
- **Tveganja, povezana z ranljivostmi v pametnih pogodbah:** Čeprav pametne pogodbe avtomatizirajo transakcije, lahko napake v njihovem kodiranju povzročijo finančno izgubo.
- **Manj zaščite pred predpisi:** Decentralizirane platforme nimajo jasnih pravnih predpisov, kar lahko oteži zaščito pravic uporabnikov v primerjavi s centraliziranimi menjalnicami.

2.3.2 Posojanje in izposoja sredstev

V nasprotju s centraliziranimi platformami ali tradicionalnimi bankami decentralizirani finančni trgi (DeFi) uporabnikom omogočajo, da deponirajo sredstva v likvidnostne sklade in prejemajo obresti ali si izposodijo sredstva proti zavarovanju (Schär, 2021, str. 12–13).

Pametne pogodbe upravljajo (Schär, 2021, str. 12–13):

- obrestne mere (na podlagi ponudbe in povpraševanja);
- razmerje posojilo/vrednost (LTV);
- likvidacije v primeru padca vrednosti zavarovanja.

Za pravilno delovanje sistema so ključni oraklji (*price oracles*), ki zagotavljajo zunanje podatke o cenah sredstev. Če orakelj ni zanesljiv, lahko pride do manipulacije cen in posledičnih likvidacij (Schär, 2021, str. 11).

Varnostno tveganje predstavljajo tudi *flash* posojila, ki omogočajo izposojo velikih zneskov brez zavarovanja, če so vrnjeni v isti transakciji (Schär, 2021, str. 12). Tak mehanizem lahko negativno vpliva na DeFi aplikacijo in njen ekosistem.

Čeprav popolna implementacija posojilnega sistema presega obseg osnovne aplikacije v tej nalogi, razumevanje teh mehanizmov predstavlja pomembno teoretično osnovo za prihodnje nadgradnje sistema.

2.3.3 Staking in yield farming

Staking

Staking temelji na mehanizmu soglasja *Proof-of-Stake* (PoS), kjer validatorji zaklenejo sredstva za zagotavljanje varnosti omrežja in v zameno prejemajo nagrade (Hacken, 2025).

V kontekstu DeFi *staking* pogosto pomeni zaklepanje žetonov v pametni pogodbi za določeno obdobje, v zameno za nagrade. To predstavlja pasivni model generiranja donosa.

Ta postopek dosega dva cilja (Hacken, 2025):

- Lastnikom žetonov daje možnost pasivnega ustvarjanja dobička iz svojih kriptovalut. To je ključnega pomena za ohranjanje integritete in varnosti omrežij veriženja blokov, ki uporabljajo metodo soglasja Proof of Stake (PoS).
- DeFi *staking* predstavlja preglednejšo in decentralizirano obliko konvencionalnih finančnih dejavnosti, kot je prejemanje dividend od delnic ali obresti na varčevalnih računih. V skladu s temeljnimi idejami decentralizacije in vključevanja skupnosti, ki so značilne za ekosistem DeFi, omogoča tudi neposredno sodelovanje uporabnikov pri delovanju in upravljanju omrežja.

Delovanje *Stakinga*

Da bi si najlažje predstavili delovanje *Stakinga*, ga razdelimo na različne mehanizme, ki jih lahko razčlenimo na naslednje načine (Hacken, 2025):

- **Dokazilo o deležu (PoS):** Temeljni postopek za doseganje soglasja se imenuje dokazilo o deležu. Količina kriptovalute, ki jo validatorji želijo vložiti kot zavarovanje, določa katere validatorje PoS izbere.
- **Validatorji:** So udeleženci v omrežju z vloženimi žetoni, njihove naloge vključujejo ustvarjanje novih blokov in preverjanje transakcij. Za slabo vedenje so lahko kaznovani, lahko pa za svoje storitve prejmejo tudi nagrade.

***Staking* postopek:** Uporabniki uporabijo primerno denarnico ali platformo za vlaganje, da zaklenejo določeno količino na primer Bitcoinov. Vloženi žetoni se nato, bodisi neposredno bodisi prek sklada za vlaganje, uporabijo za podporo delovanja omrežja. Vlagatelji prejmejo nagrade glede na svoj delež, ko omrežje obdeluje transakcije in ustvarja nove bloke.

Yield farming

Ena od strategij, ki lastnikom kriptovalut omogoča dobiček, je donosno kmetovanje ali *yield farming*, ki vključuje zagotavljanje likvidnosti protokolom decentraliziranega financiranja (DeFi). Omogoča učinkovito menjavo žetonov in ponuja možnosti za pasivni prihodek, zaradi česar je bistvena sestavina ekosistema DeFi (Kraken Learn Team, 2025).

Z donosnim kmetovanjem lahko lastniki kriptovalut zagotavljajo likvidnost protokolom DeFi v zameno za nagrade, s čimer ustvarjajo pasivni dohodek. Prispevek virov decentraliziranim platformam pomaga pri finančnih transakcijah ekosistema in pri izmenjavi žetonov (Kraken Learn Team, 2025).

Ponudniki likvidnosti (LP) so nagrajeni z žetoni LP, ki ustrezajo njihovemu sorazmernemu deležu v skladu. Vsakič, ko se sredstva izmenjajo znotraj sklada, protokol zaračuna stroške. Nato imetniki žetonov LP prejmejo te provizije kot nagrade. Višina deleža vsakega imetnika je določena glede na to, koliko je prispeval v sklad. Žetoni likvidnostnega sklada (LP) so vrsta digitalnega sredstva, ki predstavlja delež posameznika v likvidnostnem skladu DeFi (Kraken Learn Team, 2025).

Proces *yield farminga*

V procesu donosnega kmetovanja obstaja več korakov (Kraken Learn Team, 2025):

- **Izbira platforme:** Izbira platforme za DeFi, ki zagotavlja potrebne žetone za zagotavljanje likvidnosti in podpira donosno kmetovanje.
- **Zagotovitev likvidnosti:** V likvidnostni sklad moramo vstaviti dva žetona kriptovalute.
- **Pridobitev žetonov LP:** Naš delež sklada bodo predstavljali žetoni LP, ki jih bomo prejeli ob pologu. Če želimo ustvariti sekundarni dohodek, lahko te žetone vložimo na druge platforme.
- **Vlaganje in donos:** Za vlaganje naših LP žetonov uporabimo isto ali drugo platformo, ki omogoča *yield farming*. Ta taktika prinese spodbude za vlaganje v obliki dodatnih žetonov. Obrestne mere so na platformah *yield farming* pogosto prikazane kot letni odstotek donosa (APY).
- **Uveljavljanje in ponovno vlaganje:** Za optimizacijo dobička *yield farmers* zahtevajo nagrade, ki se pripišejo naši povezani denarnici Web3 in jih ponovno investirajo, da bi povečali donose.

Prednosti *yield farminga*

Za imetnike kriptovalut donosno kmetovanje ponuja privlačno priložnost za dobiček. V zameno za zagotavljanje likvidnosti platformam DeFi lahko udeleženci prejmejo ugodnosti in pomoč pri širitvi DeFi (Kraken Learn Team, 2025):

- **Prejemanje nagrad:** Z dodajanjem likvidnosti platformam DeFi nam donosno kmetovanje omogoča prejemanje nagrad. Za izplačilo nagrad se pogosto uporabljajo provizije za transakcije in žetoni za upravljanje.
- **Dostop do različnih spodbud:** Za spodbujanje zagotavljanja likvidnosti in decentralizacijo nadzora nad platformo nekatere platforme ponujajo dodatne spodbude, kot so žetoni za upravljanje.
- **Možnosti za rast kapitala:** Pridobivanje več žetonov z donosnim kmetovanjem nam omogoča povečanje naših kriptovalut. Sčasoma lahko svoje premoženje močno povečamo z reinvestiranjem dobička.

Slabosti *yield farminga*

Čeprav ima donosno kmetovanje potencial za velike dobičke, je ključnega pomena poznavati njegova tveganja oziroma Slabosti (Kraken Learn Team, 2025):

- **Začasna izguba:** Ko se vrednost sredstev, vloženih v likvidnostni sklad, precej razlikuje od njihove trenutne tržne cene, se to imenuje začasna izguba. Donosi so lahko zaradi tega manjši, kot če bi ponudnik likvidnosti samo obdržal svojo kriptovaluto. Ker se cene celoten čas spreminjajo, lahko zaslužimo manj, kot če jih ne bi vložili v likvidnostni sklad.
- **Izkoriščanje pametnih pogodb:** Čeprav so pametne pogodbe temelj DeFi, lahko vsebujejo pomanjkljivosti ali slabosti, ki bi jih napadalci lahko izkoristili. To lahko povzroči finančno izgubo ali manipulacijo z nagradami.
- **Rug pulls:** Ko je na voljo dovolj kapitala, nekateri nepošteni razvijalci izmišljujejo sheme za privabljanje likvidnosti, le da bi nato ukradli denar. Posledično sta pri odločanju, kam investirati, ključnega pomena skrbnost in učenje. Zato je pred investiranjem zelo pomembno raziskati projekt, v katerega nameravamo investirati, ter preveriti njegovo zanesljivost.

Staking v primerjavi z *yield farmingom*

Med glavnimi razlikami med obema sta (Kraken Learn Team, 2025):

- **Upravljanje:** Ko je naša kriptovaluta vložena, ni treba opraviti veliko več dela. Vlaganje (*staking*) je pasivnejši pristop. Za maksimiranje dobička zahteva donosno kmetovanje aktivno upravljanje, zlasti pri spreminjanju pozicij ali iskanju novih platform za ustvarjanje donosa. Vlaganje kriptovalut je mirnejši pristop in ne zahteva veliko dela, medtem ko *yield farming* zahteva več pozornosti in prilagajanja.
- **Donosi:** Čeprav ima donosno kmetovanje običajno večje tveganje izgube, včasih prinaša več donosov, zlasti v obliki provizij in žetonov za upravljanje. Nagrade za vlaganje so lahko stabilnejše in predvidljive.

2.3.4 DAO – decentralizirana avtonomna organizacija

Pomen DAO

DAO predstavlja model decentraliziranega upravljanja, kjer se odločitve sprejemajo prek glasovanja imetnikov žetonov. Glasovanje se izvaja prek pametnih pogodb, ki avtomatizirajo izvajanje potrjenih predlogov (Reiff, 2025).

Delovanje DAO

Ker so digitalne valute decentralizirane, jih ne nadzoruje noben posamezen subjekt, kot je vlada ali centralna banka. Namesto tega so razpršene po več vozliščih, omrežjih in računalnikih. To je motiviralo ekipo razvijalcev, ki so leta 2016 ustvarili koncept decentralizirane avtonomne organizacije. Z združeno skupino vodij in udeležencev, ki bi delovali kot vodstveni organ, je bil namen spodbujati spremljanje in upravljanje, podobno kot v korporaciji (Reiff, 2025).

Pametne pogodbe so potrebne za delovanje DAO. Ko je pridobljeno potrebno število glasov, ti programi običajno avtomatizirajo odločitve skupine. Pametna pogodba ne stori ničesar, če je predlog zavrnjen. Kot primer si zamislimo kriptovaluto z DAO, ki ima člane, ki želijo spremeniti tokenomiko blockchaina. To lahko vključuje uničenje nekaterih rezervnih žetonov, nagrajevanje trenutnih imetnikov žetonov ali povečanje količine valute v obtoku (Reiff, 2025). Pametne pogodbe v DAO torej samodejno uresničijo skupne odločitve članov, če ta isti predlog dobi dovolj glasov, drugače pa pametne pogodbe ostanejo neaktivne.

Tokenomika je preučevanje in analiza ekonomije projekta kriptovalute ali blockchaina, s poudarkom na ustvarjanju in dodelitvi njegovih izvornih digitalnih žetonov. Tokenomika predstavlja ekonomski model digitalnih žetonov (Saykhov idr., 2023). Gre za združitev ekonomskih pravil in funkcije žetona v enem konceptu.

Predlog, ki bi bil posredovan članom z glasovalno pravico, bi lahko člani dali na glasovanje. Glasovanje bi se štelo prek pametne pogodbe. Ker bi to vključevalo spremembo kode blockchaina, bi se to lahko avtomatiziralo. V vsakem primeru bi rezultat pomenil nadaljnjo pot blockchaina. Pametna pogodba bi lahko avtomatizirala prenos žetonov subjektom, vključenim v projekt, če bi glasovanje zadevalo uporabo žetonov državne blagajne za ta projekt (Reiff, 2025). Tak sistem omogoča pregledno, decentralizirano upravljanje javnih sredstev na podlagi skupnostnih odločitev.

Glasovalna moč uporabnikov se pogosto dodeli glede na to, koliko žetonov imajo v lasti. Uporabnik s 100 žetoni DAO ima lahko na primer dvakrat večjo glasovalno moč kot uporabnik s 50 žetoni (Reiff, 2025).

Prednosti DAO (Reiff, 2025):

- **Decentralizacija:** Namesto centralne oblasti se odločitve, ki vplivajo na organizacijo, sprejemajo s skupino ljudi. DAO omogoča porazdelitev pooblastil med širšim krogom uporabnikov in ni odvisen od odločitev ene same osebe.
- **Olajševanje sodelovanja:** Če vsem v organizaciji omogočimo neposreden glas in možnost glasovanja o katerem koli vprašanju, se lahko uporabniki počutijo bolj povezane in polnomočne. DAO imetnikom žetonov omogoča, da glasujejo, tudi če nimajo veliko glasovalne moči.
- **Glasovi so vidni javnosti:** Glasovi se oddajo prek tehnologije veriženja blokov znotraj DAO in so vidni javnosti. Ker so glasovi in odločitve uporabnikov vidni svetu, morajo ravnati na načine, za katere menijo, da so najboljši.
- **Spodbujanje skupnosti:** Ideja DAO ima potencial, da združi z vsega sveta, da bi ustvarili skupni cilj. Imetniki žetonov lahko komunicirajo z drugimi lastniki, od koder koli po svetu samo z internetno povezavo.

Slabosti DAO (Reiff, 2025):

- **Zamudno glasovanje:** Če javno podjetje vodi generalni direktor, lahko izbira določenega ukrepa zahteva le en glas. DAO omogoča vsem uporabnikom, da oddajo glas. Glede na časovne pasove in prioritete zunaj DAO lahko to zahteva nekoliko daljšo sejo glasovanja.
- **Postopek je treba razložiti članom:** Člani morajo vedeti o prihajajočih ukrepih DAO. Zaradi možnosti različnih izobrazbenih okolij, razumevanja ciljev, spodbud ali dostopnosti virov je za enega samega generalnega direktorja precej preprosteje odločati o rasti podjetja.

DAO združujejo raznoliko skupino ljudi, vendar je za te posameznike lahko težko razviti svoje komunikacijske, rastne ali strateške sposobnosti.

- **Neučinkovitost:** DAO-ji so v veliki nevarnosti, da postanejo neučinkoviti. Za DAO je preprosto porabiti veliko več časa za govorjenje o spremembah kot za njihovo dejansko izvedbo zaradi časa, ki je potreben za obveščanje volivcev, posredovanje ciljev, razlago metod in vpis novih članov. Ker mora DAO koordinirati veliko več ljudi, se lahko znajde v administrativnih, vsakdanjih opravilih.
- **Varnostne težave:** Izvajanje DAO zahteva visoko raven tehnične usposobljenosti. Odločanje in glasovanje bi lahko bila brez nje ogrožena. Če uporabniki ne morejo zaupati strukturi subjekta, se lahko zaupanje izda in se lahko odložijo za odhod. Zakladniški depoziti se lahko zasežejo, trezorji se lahko izpraznijo, DAO-ji pa se lahko izkoriščajo tudi pri uporabi denarnic z več podpisi ali hladnih denarnic.

Delovanje DAO v DeFi

Da bi vzpostavili DAO v DeFi ekosistemu, moramo najprej opraviti določene korake, ki bi zagotovili uspešno implementacijo in delovanje. Najprej moramo določiti namen in cilj svojega DAO, izbrati pravo platformo veriženja blokov, ki bi podpirala funkcionalnosti DAO. Nato moramo oblikovati upravljanje in strukturo DAO, kar pomeni, da se moramo odločiti za model upravljanja. Nato sledi ustvarjanje in izdajanja žetonov DAO, kar pomeni celotno razvijanje strategije tokenomike. Pametna pogodba je tudi glavna točka, saj ureja delovanja našega DAO. Pogodbe avtomatizirajo procese, kot so glasovanje, izvajanje predlogov in upravljanje sredstev. To je zelo pomemben korak z vidika varnosti in zanesljivosti DAO. Zadnji korak je uvedba in promocija DAO, ki vključuje pripravo skupnosti, izvedbo prodaje žetonov in vzpostavitev začetnih mehanizmov upravljanja. Ko imamo DAO vzpostavljen, moramo redno upravljati procese upravljanja, in nenehno razvijati in izboljševati njegove funkcije, da bo DAO ostajal učinkovit (NadcabAdmin, 2025).

2.4 Tehnološki sklad spletnih DeFi aplikacij (Solidity, Web3, Javascript, Node.js, ipd.)

Razvoj DeFi spletne aplikacije zahteva tehnološki sklad, ki omogoča izvajanje poslovne logike na blockchainu, varno obdelavo transakcij ter ustrezno uporabniško izkušnjo. V nasprotju s klasičnimi spletnimi aplikacijami, kjer sta poslovna logika in podatkovna baza centralizirani,

DeFi aplikacije temeljijo na pametnih pogodbah, ki se izvajajo na blockchain omrežju (Bomberbot, 2024). Takšna arhitektura omogoča transparentnost in nespremenljivost podatkov, vendar hkrati zahteva visoko stopnjo varnostne discipline.

V okviru diplomskega dela je bil kot osnovna infrastruktura izbran Ethereum, saj predstavlja najrazvitejši ekosistem za razvoj pametnih pogodb in decentraliziranih aplikacij. Ethereum omogoča izvajanje kode v okolju Ethereum Virtual Machine (EVM) ter podpira standarde, kot je ERC-20, ki zagotavlja interoperabilnost žetonov med različnimi protokoli (Schär, 2021, str. 5–6; Anglen, n. d.). Izbira tega omrežja je smiselna tudi zaradi obsežne dokumentacije, varnostnih smernic in orodij za testiranje, ki so na voljo za Solidity razvoj (Anglen, n. d.).

2.4.1 Veriga blokov (blockchain) plast in pametne pogodbe

Jedro DeFi spletne aplikacija predstavlja blockchain omrežje Ethereum, ki omogoča izvajanje pametnih pogodb prek Ethereum Virtual Machine (EVM). Pametna pogodba je program, ki se izvaja na blockchainu in samodejno opravlja vnaprej določena pravila brez posrednikov. V kontekstu decentraliziranih financ predstavljajo temelj decentraliziranih borz, sistemov posojanja, *staking* mehanizmov in likvidnostnih bazenov (Schär, 2021, str. 3–4).

Za razvoj pametnih pogodb se uporablja programski jezik Solidity, ki je standard v Ethereum ekosistemu in je posebej zasnovan za EVM (OSL, 2025). Uporaba standardov, kot je ERC-20, omogoča interoperabilnost žetonov med različnimi protokoli in aplikacijami (Anglen, n. d.; Reiff, 2025). Interoperabilnost v smislu, da žetoni sledijo standardu ERC-20, delujejo med seboj združljivo.

Posebna pozornost je namenjena varnostnim vzorcem, kot je načelo »*check-effect-interaction*«, ki določa, da mora pogodba najprej preveriti pogoje (*check*), nato posodobiti stanje (*effects*) in šele nato izvesti zunanjo interakcijo (*interactions*). Ta vzorec zmanjšuje tveganje *reentrancy* napadov, pri katerih napadalec večkrat vstopi v funkcijo, preden se stanje pravilno posodobi. (Marchesi idr., 2020, str. 5). Uporaba preverjenih knjižnic, kot je OpenZeppelin, dodatno zmanjšuje varnostna tveganja. (Orb, 2022)

2.4.2 Integracijska plast (Ether.js, denarnice, RPC)

Uporabniški vmesnik komunicira z blockchain omrežjem prek knjižnice Ether.js, ki omogoča klice funkcij pametnih pogodb in pošiljanje transakcij. Transakcije uporabniki potrjujejo prek

kripto denarnic (npr. MetaMask), kjer se izvajajo digitalni podpisi z zasebnimi ključi. Aplikacija sama ne upravlja zasebnih ključev, kar predstavlja pomembno varnostno načelo decentraliziranih sistemov (Gate, n. d.).

Za povezavo z omrežjem se uporabljajo ponudniki RPC (Remote Procedure Call), ki omogočajo komunikacijo z vozlišči Ethereum omrežja brez potrebe po lastnem vozlišču. Ta plast zagotavlja stabilno delovanje aplikacije, zato je pomembno preverjanje omrežja in pravilnih naslovov pametnih pogodb (CointTracker, n. d.).

Pri decentralizirani izmenjavi ima pomembno vlogo tudi koncept *slippage* (zdrs cene), ki predstavlja razliko med pričakovano in dejansko ceno izvedene transakcije zaradi sprememb razmerja v likvidnostnem bazenu (Hayes, 2025). Uporabniku mora biti omogočena nastavitev tolerance zdrsa, kar predstavlja zaščitni mehanizem pred neugodnimi cenovnimi spremembami.

2.4.3 Uporabniški vmesniki (*Frontend – React + TypeScript*)

Za razvoj uporabniškega vmesnika je izbrana kombinacija *React* in *TypeScript*, ki omogočata modularno zasnovo, boljšo tipno varnost in lažje vzdrževanje kode (Kramer, 2025). Takšna arhitektura omogoča jasno ločitev funkcionalnih modulov, kot so decentralizirana izmenjava (DEX), upravljanje likvidnosti, *staking* ter *yield farming*.

Pri zasnovi uporabniške izkušnje je ključna preglednost. Uporabniku morajo biti jasno prikazani stroški transakcije (*gas fee*), toleranca zdrsa cene *(slippage*), minimalni prejeti znesek ter status transakcije. Ker so Defi aplikacije pogosto tarča napadov na uporabniški vmesnik, je treba zagotoviti zaščito pred manipulacijo prikaza podatkov in zlonamernimi preusmeritvami transakcij.

2.4.4 Backend kot podporni sloj

Blockchain sam po sebi deluje kot porazdeljena in nespremenljiva podatkovna plast, zato jedro poslovne logike ne potrebuje klasične relacijske podatkovne baze (Bomberbot, 2024). Kljub temu je mogoče uporabiti tudi zaledni sloj (*backend*), razvit v okolju Node.js in TypeScript, za izvajanje podpornih funkcij, kot so agregacija podatkov ali integracija z zunanjimi storitvami (Beyondi, 2024).

V okviru tega diplomskega dela ostaja jedro sistema decentralizirano, medtem ko je *backend* obravnavan kot opcijska nadgradnja za izboljšanje uporabniške izkušnje in razširljivosti sistema.

2.4.5 Testno okolje in razvojna orodja

Za razvoj in preverjanje delovanja pametnih pogodb se uporabljajo razvojna orodja, kot sta Hardhat ali Ganache, ki omogočata lokalno simulacijo blockchain omrežja (Meegle, 2025). Testiranje vključuje enotske teste in preverjanje robnih primerov (*edge cases*), kar zmanjšuje možnost napak v poslovni logiki.

Za dodatno varnostno preverjanje se uporabljajo varnostna orodja za statično analizo kode, kot sta Slither in Mythril, ki omogočata odkrivanje ranljivosti pred uvedbo v produkcijo (Orb, 2022). Avtomatizacija in uvedba CI/CD postopkov dodatno povečujeta zanesljivost sistema (Beyondi, 2024).

3 KIBERNETSKA VARNOST V DEFI SPLETNIH APLIKACIJAH

Decentralizirane finance predstavljajo enega najbolj dinamičnih in hkrati najbolj tveganih segmentov blockchain ekosistema. Ker DeFi aplikacije upravljajo z digitalnimi sredstvi visoke vrednosti in temeljijo na nespremenljivih pametnih pogodbah, so varnostne napake pogosto nepovratne in vodijo do neposrednih finančnih izgub. V nasprotju s tradicionalnimi sistemi v DeFi okolju ni centralne institucije, ki bi lahko povrnila izgubljena sredstva ali razveljavila napačno izvedene transakcije (Schär, 2021, str. 16–18).

Raziskave kažejo, da so varnostne pomanjkljivosti pametnih pogodb, nepravilna integracija z zunanjimi sistemi ter ranljivosti uporabniškega vmesnika med najpogostejšimi vzroki za incidente v DeFi ekosistemu (Marchesi idr., 2020, str. 1; Cholakov, 2025). Zato je varnost ključni element arhitekture razvite aplikacije v okviru tega diplomskega dela.

3.1 Tipične grožnje DeFi spletne aplikacije

DeFi aplikacije se soočajo s kombinacijo tehničnih in ekonomskih napadov. Med najpogostejše grožnje sodijo (Marchesi idr., 2020, str. 5):

***Reentrancy* napad**

Reentrancy ali ponovni vstop je vrsta napada, pri katerem napadalec izkoristi zunanji klic v pametni pogodbi in ponovno pokliče funkcijo, še preden se stanje pogodbe ustrezno posodobi. To lahko povzroči večkratni dvig sredstev iz istega sklada.

Eden najbolj znanih zgodovinskih primerov je napad na DAO, kjer je bila izkoriščena prav takšna ranljivost.

Zaradi takšnih primerov je danes priporočena uporaba vzorca *check-effect-interaction* in Mutex, kjer se stanje pogodbe posodobi pred izvedbo zunanjega klica.

Ranljivosti pametnih pogodb

Napake v kodi, kot so *reentrancy*, *rounding errors*, nepravilno upravljanje dostopa ali napačna logika preverjanja pogojev, lahko vodijo do izgube sredstev (Marchesi idr., 2020, str. 5–9). Ker so pametne pogodbe po objavi nespremenljive, je odprava napak pogosto nemogoča brez migracije na novo pogodbo.

Flash loan napadi

Posojila Flash omogočajo izposojo velikih količin brez zavarovanja, če so vrnjena v isti transakciji. Napadalci lahko s tem manipulirajo cene v likvidnostnih bazenih ali izkoristijo logične napake protokola (Schär, 2021, str. 12).

Napadalci lahko *flash* posojila uporabijo za (Schär, 2021, str. 12):

- manipulacijo cen v likvidnostnih bazenih;
- izkoriščanje nepravilnih cenovnih izračunov;
- sprožanje likvidacij.

Če pametna pogodba uporablja cenovne podatke brez ustreznega preverjanja ali zaščite, lahko *flash loan* povzroči izgubo sredstev.

Manipulacija cen prek orakljev

Protokoli, ki se zanašajo na zunanje podatke o cenah, so ranljivi, če oraklji niso ustrezno zaščiteni. Napačni cenovni podatki lahko sprožijo nepravilne likvidacije ali napačne izračune nagrad (Schär, 2021, str. 19).

Če napadalec manipulira ceno v likvidnostnem bazenu in pametna pogodba uporablja ta podatek kot referenco, lahko pride do napačnih likvidacij ali napačnih nagrad. Zato je priporočena uporaba decentraliziranih in preverjenih orakljev.

Z vidika aplikacije je pomembno, da uporabniku jasno prikaže tveganja, povezana z zagotavljanjem likvidnosti.

Frontend napadi

Napadi na uporabniški vmesnik vključujejo zamenjavo naslovov pametnih pogodb, manipulacijo prikaza transakcij ali *phishing* napade. *Frontend* ranljivosti v DeFi okolju so pogoste, saj uporabniki pogosto slepo zaupajo prikazanemu vmesniku (Cholakov, 2025).

Rug pull in zlonamerna logika

V nekaterih primerih razvijalci sami implementirajo funkcije, ki omogočajo izčrpanje likvidnosti ali nadzor nad sredstvi uporabnikov (Meegle, 2025).

V okviru tega diplomskega dela so navedene grožnje upoštevane pri načrtovanju pametnih pogodb in uporabniškega vmesnika, zlasti pri implementaciji DEX in *staking* mehanizmov.

3.2 Varnost pametnih pogodb (ranljivosti, revizije, orodja)

Pametne pogodbe so najkritičnejši del DeFi sistema, saj neposredno upravljajo sredstva uporabnikov. Večina varnostnih incidentov je posledica ponavljajočih se napak, ki bi bilo mogoče preprečiti z uporabo preverjenih razvojnih vzorcev (Marchesi idr., 2020, str. 1).

Najpogostejše ranljivosti (Marchesi idr., 2020, str. 5–9):

- *reentrancy* napadi;
- nepravilno upravljanje privilegijev;
- nezaščiteni zunanji klici;
- nepravilna validacija vhodnih podatkov.

Priporočene prakse so (Marchesi idr., 2020, str. 5–11)**:**

- uporaba vzorca *check-effect-interaction* in Mutex;
- omejevanje dostopa z uporabo *modifierjev*;
- uporaba preverjenih knjižnic (npr. OpenZeppelin);
- enotsko testiranje in testiranje robnih primerov.

Varnostne revizije

V produkcijskih DeFi protokolih se pogosto izvajajo zunanji varnostni pregledi (*audit*). Čeprav v okviru tega diplomskega dela ni izvedena zunanja komercialna revizija, so uporabljena razvojna orodja in smernice za preverjanje varnosti kode (Marchesi idr., 2020, str. 8).

Orodja in prakse za zagotavljanje varnosti:

- statična analiza kode (Marchesi idr., 2020, str. 2);
- testna okolja (Meegle, 2025);
- uporaba standardov in uveljavljenih praks, povezanih s platformo (Marchesi idr., 2020, str. 9).

Pri razviti aplikaciji bo varnost pametnih pogodb zagotovljena z uporabo navedenih praks ter s sistematičnim testiranjem pred objavo na omrežju.

3.3 Varnost spletnega vmesnika (frontend) in zaščita uporabnika

V DeFi aplikacijah uporabniški vmesnik ni samo prikaz podatkov, ampak tudi predstavlja ključni varnostni sloj, saj uporabnika vodi skozi povezavo denarnice, izbiro funkcij DeFi aplikacije in potrditev transakcij. Ker uporabnik transakcije potrjuje s podpisom v denarnici, zavajajoč *frontend* lahko povzroči izgubo sredstev tudi v primeru, ko so pametne pogodbe tehnično pravilno implementirane. Napadi na DeFi *frontende* so pogosti, ker napadalci pogosto ciljajo najšibkejši člen – uporabniško zaupanje in infrastrukturo dostave spletne aplikacije (Cholakov, 2025).

Tipične grožnje na frontendu

Phishing in »clone« strani

Napadalci pogosto ustvarijo kopijo legitimne DeFi strani (enaka podoba) in uporabnika prepričajo, da poveže denarnico ter podpiše transakcijo ali odobritev, ki omogoči prenos sredstev. V praksi to pomeni, da uporabnik misli, da potrjuje *swap*, dejansko pa potrjuje dovoljenje ali transakcijo na napadalčev naslov (Cholakov, 2025).

Ugrabitev DNS zapisov

Tudi če so pametne pogodbe varne, *frontend* ni. Napadalci lahko pridobijo nepooblaščen dostop do zapisov DNS, kjer lahko spremenijo domeno in jo usmerijo na zlonamerni strežnik (Cholakov, 2025).

Kompromitacija dobavne verige (supply-chain)

Frontend pogosto uporablja zunanje knjižnice. Če je kompromitiran paket ali *build* proces, se lahko zlonamerna koda razširi na uporabnike (Petruk, 2023).

XSS (Cross-Site Scripting) in vnosni napadi

XSS omogoča vbrizgavanje zlonamerne skripte v spletno aplikacijo (npr. prek nefiltriranih vhodov), kar lahko spremeni prikaz transakcije ali preusmeri uporabnika (Petruk, 2023).

Prakse zaščite uporabnika v DeFi *frontendu*

Jasen prikaz transakcije, kar pomeni, da uporabniku mora biti pred podpisom jasno prikazano (Cholakov, 2025; Petruk, 2023):

- katera funkcija se kliče;
- na kateri pogodbi;

- kateri *tokeni* in zneski so vključeni;
- kateri naslov prejema sredstva;
- opozorilo pri »*approval*« transakcijah. To zmanjšuje tveganje, kjer uporabnik podpiše nekaj, česar ne razume.

Preverjanje omrežja (*chainId*) in »*wrong network*« zaščita

Frontend mora zaznati, ali je uporabnik na pravem omrežju (npr. Ethereum) in preprečiti izvajanje transakcij na napačnem *chainu.* To je pomembno za UX (User Experience) kot za varnost, saj napadalci včasih izrabijo zmedo okoli omrežij (Petruk, 2023).

***Slippage* opozorila**

V DEX okolju mora uporabnik nastaviti toleranco *slippage*, kar pomeni prenizka toleranca lahko povzroči neuspešno transakcijo, previsoka toleranca pa poveča tveganje neugodnega izvrševanja (Petruk, 2023). *Frontend* mora *slippage* jasno razložiti in uporabnika opozoriti pri tveganih nastavitvah.

Zmanjšane napak s TypeScript

Ker DeFi UI obravnava finančne podatke, je zmanjšanje tipičnih programskih napak pomembno. Odločil sem se za uporabo TypeScripta, saj pomaga pri tipizaciji in zmanjšanju »runtime« napak pri obdelavi podatkov, kar prispeva k zanesljivosti integracije (Nundini, 2025; Kramer, 2025).

3.4 Varnost komunikacije med frontend – smart contracts – backend

DeFi aplikacija ima več komunikacijskih poti:

- frontend – blockchain vozlišča (prek RPC);
- frontend – kripto denarnica;
- frontend – backend API.

Frontend – Blockchain (komunikacija RPC)

RPC (*Remote Procedure Call*) omogoča, da aplikacija pošilja zahteve vozlišču blockchain omrežja (npr. za branje stanja ali pošiljanje transakcij), ne da bi morala upravljati lastno vozlišče (*node*) (CoinTracker, n. d.).

Tipična tveganja:

- uporaba napačnih ali zastarelih podatkov o stanju sistema;

- nedosegljivost oziroma nedelovanje (*downtime*) ponudnika RPC-storitev;
- nepravilna konfiguracija končnih točk (*endpoints*);
- možnost napadov tipa »*man-in-the-middle*« zaradi neustrezne konfiguracije, pomanjkljive zaščite komunikacijskih kanalov ali kompromitacije infrastrukture.

Zaščitne prakse:

- preverjanje *chainId* pred kritičnimi akcijami;
- poraba preverjenih RPC-ponudnikov;
- *fallback* RPC;
- robustna obravnava napak in ponovni poizkusi;
- primerjanje kritičnih podatkov prek več virov.

RPC, *fallback* RPC in preverjeni RPC-ponudniki so trije med seboj povezani koncepti, ki skupaj zagotavljajo zanesljivo komunikacijo med DeFi aplikacijo in blockchainom.

RPC je v osnovi mehanizem, s katerim aplikacija vpraša blockchain, naj izvede določeno funkcijo in vrne rezultat. RPC omogoča komunikacijo med dvema sistemoma, kjer ena aplikacija zahteva izvedbo funkcije na drugi aplikaciji in prejme rezultat. V praksi pri DeFi aplikacijah so RPC klici klik v denarnici, zamenjava žetonov, vsaka funkcija, ki se izvede v ekosistemu (Ribhavmodi, 2025).

Preverjeni RPC-ponudniki so storitve, ki izpostavljajo metode blockchain vozlišč prek HTTPS ali WebSocketa, tako da razvijalcem ni treba poganjati lastnih vozlišč. Vzdrževanje lastnih vozlišč je namreč drago in zahteva infrastrukturo v oblaku in prinaša tveganje izpadov zaradi CPU-konic, uhajanja pomnilnika ali težav z diskom (DWELLIR, 2025).

Fallback RPC pa je mehanizem za preklop ob odpovedi, lahko je samodejni ali ročni. Gre za rezervno komunikacijsko pot v primeru, če RPC ponudnik odpove ali je nedosegljiv, aplikacija samodejno preklopi na alternativnega in nemoteno nadaljuje delovanje. Ena pogostejših napak pri razvoju je trdo vkodiran (*hardcoded*) enoten ponudnik brez alternative (*fallbacka*) (DWELLIR, 2025).

***Frontend* – denarnica (podpisovanje)**

Kripto denarnica izvaja podpisovanje lokalno, zato aplikacija ne upravlja zasebnih ključev, to je varnostna prednost. Vendar je ključni problem, »kaj uporabnik podpiše«. Če *frontend* zavaja ali če uporabnik ne razume *approval* transakcij, lahko pride do zlorabe (Cholakov, 2025; Bomberbot, 2024).

***Frontend – Backend* (če obstaja podporni sloj)**

Backend v decentralizirani aplikaciji predstavlja podporni sloj, ki omogoča dodatne funkcionalnosti, kot so analitika, agregacija podatkov in obvestila. Vendar pa ne predstavlja osrednje točke zaupanja, saj se ključna poslovna logika in stanje hranita na blockchainu.

3.5 Pripomočki in prakse za zagotavljanje kibernetske varnosti

Varnost DeFi aplikacije je kombinacija varnosti pametnih pogodb, varnost *frontenda*, varnost infrastrukture in procesi. Pri omrežju Ethereum je smiselno slediti varnostnim »*checklistam*« in standardiziranim postopkom, ker se ranljivosti pogosto ponavljajo (Marchesi idr., 2020, str. 1–2).

Prakse pri razvoju pametnih pogodb (Marchesi idr., 2020, str. 5–9; Orb, 2022):

- uporaba preverjenih knjižnic (npr. OpenZeppelin) in vzorcev;
- enotsko testiranje in *edge-case* testiranje.
- uporaba statične analize in samodejnih pregledov;
- »*least privilege*« pri dostopu;
- varna obravnava zunanjih klicev in uporaba *check-effect-interaction*.

Prakse pri razvoju frontenda (Petruk, 2023; Cholakov, 2025):

- zaščita pred XSS, pravilna validacija vhodov;
- prikaz »*what you sign*«;
- preverjanje omrežja in naslovov pogodb;
- varna uporaba odvisnosti.

Prakse v CI/CD in infrastrukturi

Backend in deployment procesi pogosto predstavljajo napadalno površino, zato so pomembni (Beyondi, 2024):

- avtomatizirani testi v CI;
- skeniranje ranljivosti odvisnosti;
- ločevanje okolij;
- varno upravljanje skrivnosti in ključev.

Varnost DeFi je večplastna in mora zajemati tako kodo kot tudi operativne prakse.

3.6 Primer vdorov v DeFi spletne aplikacije in analize posledic

Analiza realnih incidentov je pomembna, ker pokaže, da so napadi pogosto posledica kombinacije tehničnih in procesnih napak. DeFi protokoli delujejo v medsebojno povezanem ekosistemu, kar poveča sistemsko tveganje. To pomeni, da lahko napaka v enem protokolu vpliva tudi na druge.

Primer: DAO (*reentrancy*)

DAO incident je klasičen primer *reentrancy* ranljivosti, kjer je napadalec izkoristil zunanji klic in ponavljajoče se dvige pred posodobitvijo stanja. Posledice so bile tako velike, da je prišlo do razkola v skupnosti in *hard forka*, kar kaže, da imajo varnostni incidenti v DeFi lahko tudi širše ekosistemske posledice (Marchesi idr., 2020, str. 5).

Primeri *frontend* vdorov

Takšni napadi so prek kompromitiranega *frontenda*, npr. zamenjava naslovov ali zlonamerni prikaz transakcij. Posledica takih napadov je, da uporabnik sam potrdi transakcijo, zato je povračilo sredstev praktično nemogoče (Cholakov, 2025).

Sklep analize posledic

Skupna značilnost incidentov je:

- pomanjkljivo testiranje ali *review*;
- neuporaba preverjenih varnostnih vzorcev;
- slaba zaščita uporabniškega vmesnika;
- pomanjkanje varnostnih procesov.

4 NAČRTOVANJE SPLETNE APLIKACIJE

4.1 Definicija funkcionalnih zahtev za spletno DeFi rešitev

Načrtovanje decentralizirane finančne aplikacije se začne z jasno opredelitvijo funkcionalnih zahtev. Funkcionalne zahteve določajo, katere operacije mora sistem omogočati uporabniku ter kako se te operacije izvajajo v okviru blockchain infrastrukture.

V okviru diplomskega dela aplikacija vključuje ključne funkcionalnosti, ki so navedene v nadaljevanju.

Povezava z denarnico

Uporabnik mora imeti možnost povezati svojo kripto denarnico s sistemom. Sistem pa mora:

- preveriti pravilno omrežje (*chainId*);
- pridobiti naslov uporabnika;
- omogočiti odklop povezave.

Ta funkcionalnost je osnovni pogoj za vse nadaljnje interakcije z DeFi protokolom.

Decentralizirana izmenjava (DEX)

Aplikacija mora omogočati:

- izbiro para žetonov;
- prikaz trenutnega razmerja v likvidnostnem bazenu;
- izračun pričakovane količine prejetega žetona;
- nastavitev tolerance *slippage*;
- potrditev transakcije.

Pametna pogodba mora:

- upravljati likvidnostni bazen;
- izračunavati zamenjavo po AMM-modelu;
- posodabljati stanje bazena po vsaki transakciji.

Zagotavljanje likvidnosti

Uporabnik mora imeti možnost:

- deponirati par žetonov v likvidnostni bazen;
- prejeti LP-žetone;

- umakniti likvidnost.

Sistem mora pravilno izračunati delež uporabnika v skladu in sorazmerni delež provizij.

***Staking* mehanizem**

Aplikacija mora omogočati:

- zaklep določenega števila žetonov;
- spremljanje trajanja zaklepa;
- izračun nagrade;
- dvig nagrade.

Pametna pogodba mora:

- voditi evidenco vloženih sredstev;
- izračunavati nagrade glede na čas in delež;
- preprečiti večkratni dvig iste nagrade.

***Yield farming* logika**

Yield farming bo implementiran kot nadgradnja staking mehanizma, kjer:

- uporabnik prejme dodatne nagrade za zagotavljanje likvidnosti;
- sistem izračunava donose na podlagi parametrov sklada.

Ta funkcionalnost mora biti zasnovana varno, da prepreči manipulacijo z nagradami.

Varnostne zahteve

Ker aplikacija upravlja digitalna sredstva, mora sistem:

- uporabljati preverjene knjižnice;
- implementirati *check-effect-interaction*;
- preprečiti *reentrancy*;
- omejiti administrativne funkcije;
- preverjati vhodne podatke.

Nefunkcionalne zahteve

Poleg funkcionalnih zahtev so pomembne tudi nefunkcionalne zahteve:

- varnost;
- razširljivost;
- preglednost transakcij;
- zanesljivost komunikacije z omrežjem;

- odzivnost uporabniškega vmesnika.

4.2 Zasnova uporabniškega vmesnika in uporabniške izkušnje (UI/UX)

Uporabniški vmesnik decentralizirane finančne aplikacije ima bistveno vlogo, saj predstavlja povezavo oziroma most med kompleksno blockchain infrastrukturo in končnim uporabnikom. V nasprotju s klasičnimi spletnimi aplikacijami uporabnik v DeFi okolju neposredno podpisuje transakcije, ki so po potrditvi nespremenljive. Zato mora biti uporabniška izkušnja zasnovana tako, da poleg intuitivnosti zagotavlja tudi jasno predstavitev tveganj in parametrov transakcije (Cholakov, 2025).

V okviru diplomskega dela je uporabniški vmesnik zasnovan z uporabo ogrodja React in programskega jezika TypeScript, kar omogoča modularno strukturo komponent in boljši nadzor nad tipizacijo podatkov. Takšna zasnova zmanjšuje možnost logičnih napak pri obdelavi finančnih podatkov (Nundini, 2025).

Arhitekturna razdelitev uporabniškega vmesnika

Uporabniški vmesnik je razdeljen na naslednje funkcionalne module:

- modul za povezavo z denarnico;
- modul DEX (zamenjava žetonov);
- modul za zagotavljanje likvidnosti;
- *staking* modul;
- *yield farming* modul;
- nadzorna plošča (pregled sredstev, nagrad, zgodovina).

Takšna modularna zasnova omogoča ločeno upravljanje posameznih funkcionalnosti ter zmanjšuje kompleksnost posamezne komponente. Uporabnik v smislu pregled sredstev bo imel v vpogled svoj »portfolio« sredstev.

UX kot varnostni element

V DeFi aplikacijah uporabnik pogosto ne razume v celoti tehničnih podrobnosti, kot so »*approval*«, »*gas fee*«, »*slippage tolerance*« ali interakcija z različnimi pametnimi pogodbami. Zato mora uporabniški vmesnik zagotoviti naslednje:

Jasno prikazovanje transakcije pred podpisom

Pred potrditvijo mora biti uporabniku jasno prikazano:

- kateri žeton oddaja;
- kateri žeton prejema;
- količina;
- naslov pametne pogodbe;
- uporabljeno omrežje;
- ocenjeni strošek transakcije (gas).

Transparenten prikaz parametrov je ključnega pomena, saj številni napadi temeljijo na zavajanju uporabnika pri podpisovanju transakcije (Cholakov, 2025).

Upravljanje *slippage* (zdrs cene)

Slippage pomeni razliko med pričakovano in dejansko izvršeno ceno zaradi spremembe razmerja v likvidnostnem bazenu ali tržne volatilnosti. Če uporabnik nastavi previsoko toleranco, lahko transakcija poteka po bistveno slabši ceni (Hayes, 2025).

Zato mora UI:

- prikazati predvideni izračun;
- opozoriti na visoko nastavljeno toleranco;
- prikazati minimalno količino prejetega žetona.

Ločitev med »*approval*« in »*action*« transakcijo

V ERC-20 standardu mora uporabnik pogosto najprej odobriti (*approve*) porabo žetonov, preden lahko izvede *swap* ali *staking*. Uporabniki pogosto ne razumejo, da »*approval*« pomeni dovoljenje pametni pogodbi za upravljanje določenega zneska (Reiff, 2025).

UI mora jasno ločiti:

- transakcijo odobritve;
- dejansko transakcijo zamenjave ali *stakinga*.

To zmanjšuje tveganje interpretacij in zlorab.

Preverjanje omrežja

Ker aplikacija deluje na Ethereum omrežju, mora UI preveriti, ali je uporabnik povezan na pravilno omrežje (*chainId*). Če je uporabnik povezan na napačno omrežje, mora aplikacija prikazati opozorilo in onemogočiti določene funkcije, ki so kritične v tem primeru (MetaMask, n. d.).

Zmanjševanje kompleksnosti

Eden večjih izzivov DeFi aplikacij je tehnična kompleksnost. DeFi protokoli so pogosto sestavljeni iz več medsebojno povezanih slojev (»*money legos*«), kar povečuje kompleksnost sistema (Schär, 2021, str. 17).

Zato je v tej aplikaciji zasnovan minimalen in pregleden vmesnik, ki:

- ne prikazuje nepotrebnih tehničnih podrobnosti;
- poudarja ključne informacije (znesek, cena, nagrada);
- uporablja vizualne indikatorje tveganja.

Dostopnost in razširljivost

Ker je aplikacija zasnovana kot modularna *React* struktura, je možno posamezne komponente nadgrajevati ali razširjati brez posega v celoten sistem. Takšna zasnova omogoča lažje vzdrževanje in nadaljnji razvoj.

Uporabniški vmesnik je del varnostne arhitekture sistema. Transparentnost, jasna predstavitev parametrov in preverjanje omrežja predstavljajo ključne elemente uporabnika pred napakami in zlorabami.

4.3 Načrt arhitektur spletne aplikacije

Arhitektura DeFi spletne aplikacije je zasnovana kot večslojni sistem, kjer se ključna poslovna logika izvaja decentralizirano na blockchainu, uporabniški vmesnik pa omogoča varno in pregledno interakcijo uporabnika s pametnimi pogodbami. Takšna zasnova sledi tipični arhitekturi decentraliziranih aplikacij (*dApps*), kjer blockchain predstavlja »*system of record*«, uporabniški vmesnik pa deluje kot aplikacijska plast, ki posreduje med uporabnikom in protokolom (Bomberbot, 2024). Pri DeFi ekosistemu je ključna večplastna struktura – od poravnalne plasti do aplikacijske plasti (Schär, 2021, str. 3-4).

Arhitektura je zasnovana tako (Beyondi, 2024):

- decentralizirano jedro (*smart contracts*) vsebuje finančno logiko in stanje;
- *frontend* zagotavlja UX in varno podpisovanje transakcij;
- *backend* je opcijski podporni sloj (ne upravlja sredstev), namenjen izboljšanju uporabniške izkušnje in analitike.

4.3.1 Frontend (React/HTML/CSS + Web3.js/ Ether.js)

Uporabniški vmesnik je zasnovan kot spletna aplikacija (SPA), kjer *frontend* predstavlja glavno vstopno točko za uporabnika. V DeFi kontekstu *frontend* ni zgolj vizualna plast, ampak vpliva tudi na varnost, saj uporabnika vodi skozi potrjevanje transakcij in nastavitvami, kot so tolerance, *slippage* ter odobritve (*approvals*) (Cholakov, 2025; Petruk, 2023).

Ključne odgovornosti frontenda

Frontend mora omogočati:

- povezavo z denarnico (npr. MetaMask);
- prikaz stanja (*balance*, *allowance*, *staked amount*);
- sprožanje funkcionalnosti (*swap*, *add/remove liquidity*, *stake/unstake*, *claim rewards*);
- prikaz transakcij parametrov pred potrditvijo;
- preverjanje omrežja in pravilnih naslovov pogodb.

Integracija z blockchainom

Interakcija s pametnimi pogodbami poteka prek knjižnice Ether.js, ki omogoča:

- branje stanja iz pametnih pogodb (*read calls*);
- pošiljanje transakcij (*write calls*);
- dekodiranje dogodkov (*events*) za prikaz sprememb po transakcijah.

Ta pristop sledi praksam integracije DeFi protokolov v Web3 aplikacije, kjer *frontend* predstavlja aplikacijsko plast nad protokolno plastjo (Tatarchenko, 2025).

UI/UX varnostni elementi

Frontend mora v DeFi okolju dodatno implementirati zaščitne mehanizme:

- jasno ločevanje *approval* transakcij od dejanskih akcij (*swap/stake*);
- prikaz in omejitev *slippage* tolerance;
- opozorila pri tveganih nastavitvah;
- prikaz naslovov pogodb in omrežja.

Saj se napadi pogosto izvajajo prek zavajajočega vmesnika.

4.3.2 Backend (Node.js/Express, povezava z blockchain)

V DeFi aplikacijah *backend* ni nujno potreben za izvajanje finančne logike, saj so ključni podatki in poslovna logika shranjeni ter izvajani na blockchainu. Kljub temu se *backend* pogosto uporablja kot podporni sloj, ki izboljša uporabniško izkušnjo in omogoča dodatne funkcije, kot so analitika, agregacija podatkov ali integracije z zunanjimi storitvami (Bomberbot, 2024).

Vloga *backend* sloja v DeFi aplikaciji

Backend je obravnavan kot opcijska komponenta, katere namen je:

- agregacija podatkov (npr. izračun APR/APY, TVL, statistike);
- *cache* pogostih poizvedb (da se zmanjša število RPC-klicev);
- indeksiranje dogodkov (*events*) za hitrejši prikaz zgodovine;
- priprava podatkov za *dashboard*.

Pomembno je, da backend:

- ne upravlja zasebnih ključev;
- ne podpisuje transakcij;
- ne hrani kritičnega »*statea*« kot edine resnice.

Backend je zgolj kot podporna plast, saj decentraliziranost temelji na tem, da je blockchain primarni vir resnice (Bomberbot, 2024).

Varnostni vidik backenda

Backend v DeFi aplikacijah pogosto predstavlja dodatno napadalno površino, zato so potrebne prakse, kot so:

- zaščita API-jev (avtentikacija, *rate limiting*);
- varno upravljanje skrivnosti;
- segmentacija in minimalne pravice;
- spremljanje (*logging*/monitoring).

Zaradi tega je v arhitekturi aplikacije *backend* omejen na podporne funkcije in ne predstavlja centralne točke zaupanja za finančne operacije.

4.3.3 Blockchain del (pametna pogodba)

Blockchain plast predstavlja decentralizirano jedro aplikacije. Pametne pogodbe hranijo stanje (npr. likvidnost, vložki uporabnikov, nagrade) in izvajajo poslovno logiko (DEX *swap*, *staking*, *yield farming*). DeFi protokoli temeljijo na pametnih pogodbah kot avtomatiziranih finančnih pravilih, kar omogoča delovanje brez posrednikov (Schär, 2021, str. 1–2).

Ključne komponente pametnih pogodb

V okviru aplikacije so pametne pogodbe zasnovane modularno, in sicer kot:

- pogodba za upravljanje žetonov (ERC-20 ali kompatibilno);
- pogodba za DEX/likvidnostni bazen (AMM-logika);
- pogodba za *staking* (evidenca vložkov + nagrade);
- pogodba za *yield farming* (nadgradnja nagradnega mehanizma oziroma distribucija dodatnih spodbud).

Standardizirani pristopi in interoperabilnost so pomembni, saj omogočajo boljši razvoj in integracijo z drugimi komponentami ekosistema (Anglen, n. d.).

Varnost pametnih pogodb kot arhitekturna zahteva

Ker pametne pogodbe po objavi praviloma ni mogoče popravljati brez migracije, mora biti varnost vključena že v fazi načrtovanja. Uporaba varnostnih kontrolnih seznamov, testiranja robnih primerov in preverjanje vzorcev so ključni.

V tej arhitekturi se zato upoštevajo:

- *check-effect-interaction* (zmanjšanje *reentrancy* tveganja);
- omejevanje privilegijev (*owner/admin* funkcije minimalno);
- uporaba preverjenih knjižnic;
- sistematično testiranje pred objavo.

Povzetek arhitekture in tok podatkov

Arhitektura je zasnovana tako, da uporabnik prek *frontenda* sproži funkcijo, denarnica podpiše transakcijo, transakcija se izvrši na blockchainu, rezultat pa se uporabniku prikaže prek branja stanja ali dogodkov (Beyondi, 2024; Bomberbot, 2024):

- uporabnik izbere funkcijo (*swap/stake/farm*) v *frontendu*;
- *frontend* pripravi klic pametne pogodbe in prikaže parametre (npr. *slippage*, znesek);
- denarnica podpiše transakcijo;

- transakcija se pošlje v Ethereum omrežje prek RPC;
- pametna pogodba izvrši logiko in posodobi stanje;
- *frontend* osveži prikaz z branjem stanja ali dogodkov (opcijsko prek backenda).

Takšna zasnova ohrani decentralizirano jedro in zmanjšuje odvisnost od centralnih komponent, hkrati pa omogoča dobro uporabniško izkušnjo in razširljivost aplikacije (Beyondi, 2024; Bomberbot, 2024).

4.4 Poslovna logika in uporabniški primeri uporabe

Poslovna logika decentralizirane finančne aplikacije je implementirana v pametnih pogodbah na omrežju Ethereum, medtem ko uporabniški vmesnik omogoča interakcijo z njimi. V DeFi sistemih poslovna pravila niso izvedena na strežniku, temveč so trajno zapisana na blockchainu, kar zagotavlja transparentnost in avtomatizirano izvrševanje (Schär, 2021, str. 1–2). Arhitektura aplikacije je zasnovana za omrežje Ethereum, ki predstavlja osnovno poravnalno plast sistema.

V nadaljevanju pa so predstavljeni ključni uporabniški primeri uporabe (*use cases*), ki določajo tok delovanja aplikacije.

4.4.1 Uporabniški primer: povezava z denarnico

Opis:

- uporabnik obišče aplikacijo in poveže svojo kripto denarnico.

Tok dogodkov:

- uporabnik klikne na »*connect wallet*«;
- *frontend* zahteva dostop do naslova prek Web3 klica;
- denarnica zahteva potrditev povezave;
- aplikacija preveri *chainId*;
- ob uspešni povezavi se prikaže naslov uporabnika in stanje žetonov.

Varnostni vidik:

- preverjanje pravilnega omrežja;
- aplikacija nikoli ne dostopa do zasebnega ključa.

4.4.2 Uporabniški primer: Zamenjava žetonov (DEX swap)

Opis:

- uporabnik želi zamenjati žeton A za žeton B.

Logika delovanja:

- uporabnik izbere količino žetona A;
- *frontend* izračuna pričakovani izhod na podlagi razmerja v likvidnostnem bazenu (AMM-model).
- uporabnik nastavi toleranco *slippage*;
- če je potrebno, se izvede »*approval*« transakcija;
- uporabnik potrdi »*swap*« transakcijo.

Pametna pogodba:

- preveri pogoje (*check*);
- posodobi stanje bazena (*effects*);
- omejitev *slippage*;
- validacija vhodnih podatkov.

Varnostni elementi:

- *check-effect-interaction*;
- omejitev *slippage*;
- validacija vhodnih podatkov.

4.4.3 Uporabniški primer: Dodajanje likvidnosti

Opis:

- uporabnik želi zagotoviti likvidnost v likvidnostnem bazenu.

Tok dogodkov:

- uporabnik vnese količino obeh žetonov;
- sistem preveri pravilno razmerje;
- izvede se *approval* (če je potrebno).

Pametna pogodba:

- poveča stanje bazena;
- izda LP-žetone uporabniku.

Poslovna logika:

- delež uporabnika je sorazmeren vloženemu kapitalu;
- provizije od *swap* transakcij se razdelijo med LP-imetnike.

4.4.4 Uporabniški primer: staking

Opis:

- uporabnik zaklene določeno količino žetonov za prejem nagrad.

Logika delovanja:

- uporabnik vnese znesek za staking;
- izvede *approval* (če je potrebno).

Pametna pogodba:

- zabeleži vložek;
- zabeleži čas začetka;
- izračunava nagrado na podlagi časa in parametrov sklada.

Izračun nagrad

Nagrada je funkcija:

- vloženega zneska;
- trajanja vložka;
- globalne nagradne stopnje.

Varnost:

- preprečitev večkratnega dviga nagrad;
- zaščita pred manipulacijo časovnih parametrov.

4.4.5 Uporabniški primer: yield farming

Opis:

- uporabnik zagotavlja likvidnost in dodatno prejema nagrade v obliki spodbujevalnih žetonov.

Logika:

- uporabnik prejme LP-žetone;

- LP-žetone vloži v *farming* pogodbo;
- pogodba periodično izračunava dodatne nagrade;
- uporabnik lahko zahteva izplačilo nagrad.

Tveganja:

- *impermanent loss*;
- napake v izračunu nagrad;
- napačna distribucija spodbud.

4.4.6 Uporabniški primer: dvig sredstev

Opis:

- uporabnik umakne likvidnost ali prekine *staking*.

Tok:

- uporabnik sproži »*unstake*« ali »*remove liquidity*«.

Pametna pogodba:

- preveri stanje;
- izračuna nagrado;
- posodobi notranje stanje;
- vrne sredstva.

Varnostni vidik:

- zaščita pred *reentrancy*;
- preverjanje zadostnega stanja;
- pravilna obravnava nagrad.

Logična povezanost sistema

Vsi uporabniški primeri sledijo enakemu vzorcu:

Uporabnik -> Frontend -> Denarnica -> Pametna pogodba -> Blockchain -> Povratni podatki -> Frontend

Takšna arhitektura omogoča:

- decentralizirano izvrševanje;
- transparentnost;
- zmanjšanje odvisnosti od centralne infrastrukture;
- jasen nadzor nad sredstvi.

5 RAZVOJ SPLETNE DEFI APLIKACIJE

To poglavje že sodi v praktični del tega diplomskega dela, ki obsega razvoj prototipne DeFi spletne aplikacije na osnovi zahtev in arhitekturnih izhodišč, opredeljenih v prejšnjih poglavjih. Po teoretični obravnavi decentraliziranih financ, varnostnih tveganj in načrtovanja sistema je cilj praktičnega dela razviti funkcionalno rešitev, ki povezuje pametne pogodbe, uporabniški vmesnik, integracijsko plast in podporni *backend* sloj v enotno celoto.

Razvoj aplikacije temelji na omrežju Ethereum, kjer so bile ključne poslovne funkcionalnosti implementirane s pomočjo pametnih pogodb v jeziku Solidity. Uporabniški vmesnik je bil izdelan z uporabo ogrodja React in programskega jezika TypeScript, pri čemer je bila posebna pozornost namenjena preglednosti transakcij, pravilni komunikaciji z verigo blokov ter zaščiti uporabnika pri interakciji z decentraliziranimi funkcionalnostmi. Poleg tega je bil v aplikacijo vključen tudi *backend* sloj, razvit z uporabo okolja Node.js in programskega jezika TypeScript, katerega naloga je podpora analitičnim in pomožnim funkcijam sistema.

Praktični del vključuje razvoj pametnih pogodb, implementacijo uporabniškega vmesnika, vzpostavitev povezave med *frontendom* in blockchain omrežjem, vključitev podpornega *backend* sloja, pripravo testnega okolja ter povezovanje vseh komponent v funkcionalno celoto. Posebna pozornost je bila namenjena modularnosti arhitekture, preglednosti uporabniške interakcije, zanesljivosti komunikacije z omrežjem ter upoštevanju varnostnih načel, ki so bila obravnavana že v prejšnjih poglavjih.

5.1 Razvoj pametnih pogodb v jeziku Solidity

Pametne pogodbe predstavljajo pomemben del razvite DeFi aplikacije, saj vsebujejo poslovno logiko sistema in upravljajo stanje ključnih funkcionalnosti neposredno na blockchain omrežju. V okviru praktičnega dela so bile uporabljene kot temelj za implementacijo zamenjave žetonov, upravljanje likvidnostnega bazena, *staking* mehanizma ter *yield farming* funkcionalnosti. Takšna zasnova sledi izhodiščem iz teoretičnega dela, kjer je bilo poudarjeno, da pametne pogodbe v DeFi okolju prevzemajo vlogo avtomatiziranega izvajanja pravil brez centralnega posrednika.

Pri razvoju prototipa so bile pametne pogodbe zasnovane modularno. Namesto ene samo obsežne pogodbe so bile posamezne funkcionalnosti razdeljene na več ločenih pogodb, pri čemer vsaka pokriva jasno določeno področje sistema. Takšen pristop izboljšuje preglednost

implementacije, omogoča lažje testiranje posameznih funkcionalnosti sklopov in zmanjšuje kompleksnost poslovne logike znotraj posamezne komponente. Modularna zasnova hkrati omogoča tudi lažje nadaljnje nadgradnje sistema.

V implementirani rešitvi so bile uporabljene pogodbe za žetonsko plast, likvidnostni bazen, *staking* in *yield farming*. Med seboj so povezane funkcionalno, vendar so odgovornosti posameznih pogodb jasno razmejene. S tem je bilo mogoče doseči bolj strukturirano implementacijo, ki je skladna z arhitekturo aplikacije, predstavljeno v četrtem poglavju tega diplomskega dela.

Pri razvoju pametnih pogodb je bila pomembna tudi varnost. Ker pametne pogodbe po objavi na omrežju praviloma ni mogoče preprosto spreminjati, je bilo treba že v fazi implementacije upoštevati preverjanje vhodnih parametrov, ločevanje administrativnih funkcij, uporabo preverjenih knjižnic ter zaščito pred tipičnimi ranljivostmi DeFi okolja. S tem je praktični del neposredno povezan tudi z varnostnimi zahtevami, obravnavanimi v tretjem poglavju diplomskega dela.

5.1.1 Žetonska pogodba

Žetonska pogodba predstavlja osnovni ERC-20 sloj celotne aplikacije. Implementirana je z uporabo preverjene knjižnice OpenZeppelin, kar zmanjša tveganje napak v osnovni logiki za prenos žetonov, evidentiranje stanja in obravnavo dovoljenj. Pogodba ob ustvarjanju prejme naziv žetonov, simbol in začetno sredstvo za *staking* ter kot eno od dveh sredstev v likvidnostnem bazenu modula DEX.

Ta del implementacije se neposredno navezuje na teoretični del, zlasti na poglavje 2.4.1, kjer je poudarjena vloga standarda ERC-20 in interoperabilnosti žetonov v Ethereum ekosistemu, ter na poglavje 2.3, kjer so predstavljene funkcionalnosti, ki temeljijo na žetonski plasti. V praktičnem delu je bila zato žetonska pogodba zasnovana kot izhodišče za vse nadaljnje interakcije z aplikacijo.

Z varnostnega vidika je pri tej pogodbi pomembna predvsem uporaba preverjenih knjižnic in standardiziranih funkcionalnosti, saj se s tem zmanjšuje tveganje napak v temeljni logiki prenosa in odobritev. Ta pristop je usklajen s teoretičnim poglavjem 3.2, kjer je izpostavljena uporaba preverjenih knjižnic kot ena izmed priporočljivih praks pri razvoju pametnih pogodb.

Poleg tega je bila pozornost namenjena tudi jasni razmejitvi med standardnimi žetonskimi funkcijami in dodatno logiko drugih pogodb, kar zmanjšuje kompleksnost sistema.

Na sliki 2 je prikazan izsek implementacije žetonske pogodbe ERC20Token, ki temelji na standardu ERC-20 in ob inicializaciji ustvari začetno ponudbo žetonov.

```
pragma solidity ^0.8.28;

import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

contract ERC20Token is ERC20 {
    constructor(
        string memory name_,
        string memory symbol_,
        uint256 initialSupply_
    ) ERC20(name_, symbol_) {
        _mint(msg.sender, initialSupply_);
    }
}
```

Slika 2: Implementacija žetonske pogodbe ERC20Token na osnovi standarda ERC-20

(Vir: Lastni vir)

5.1.2 Pogodba za likvidnostni bazen

Pogodba za likvidnostni bazen predstavlja jedro modula DEX. Njena naloga je upravljanje dveh sredstev v bazenu, dodajanje in odstranjevanje likvidnosti ter izvedba menjav med žetonoma. Pogodba uporablja AMM-pristop, pri katerem se količina izhodnega sredstva določa glede na trenutno razmerje rezerv v bazenu. Ob dodajanju likvidnosti se uporabniku izda LP-žeton, ki predstavlja njegov delež v bazenu. Pri menjavi je uporabljena tudi zaščita z minimalnim sprejemljivim izhodom (»*minAmountOut*«), s čimer se omeji zdrs cene in zmanjša tveganje, da bi uporabnik prejel manj sredstev od pričakovanega.

Ta implementacija se neposredno navezuje na teoretični del, predvsem na poglavje 2.3.1, kjer je razložen koncept decentralizirane izmenjave in logika avtomatiziranega ustvarjalca trga, ter na poglavje 4.4.2 in 4.4.3, kjer sta opisana uporabniška primera zamenjave žetonov in dodajanja likvidnosti. V praktičnem delu je bila ta teorija prenesena v pogodbo, ki izvaja menjalne operacije na podlagi razmerja sredstev v likvidnostnem bazenu in upravlja spremembe stanja po vsaki transakciji.

Z varnostnega vidika je ta pogodba precej občutljiva, zato so bili pri implementaciji uporabljeni mehanizmi, ki izhajajo iz teoretičnega dela poglavja 3.2 in 3.5. Med njimi so preverjanje vhodnih podatkov, uporaba pogojev za zavrnitev neveljavnih ali tveganih transakcij, omejevanje neustreznih stanj sistema ter dosledna logika zaporedja izvajanja notranjih operacij. Pri menjalnih funkcijah je bila upoštevana tudi zaščita pred neugodnim zdrsom cene, kar je povezano s teoretično razlago *slippage* mehanizma v poglavju 2.4.2 in uporabniško-varnostnimi zahtevami iz poglavja 4.2. Takšna implementacija prispeva k bolj predvidljivemu in varnemu delovanju menjalnega sistema.

Na sliki 3 je prikazan izsek kode pametne pogodbe za likvidnostni bazen, ki vsebuje logiko za dodajanje likvidnosti in preverjanje vhodnih pogojev.

```
function _addLiquidity(
    uint256 amountADesired,
    uint256 amountBDesired,
    uint256 amountAMin,
    uint256 amountBMin
)
    internal
    returns (uint256 amountA, uint256 amountB, uint256 liquidityMinted)
{
    if (amountADesired == 0 || amountBDesired == 0) revert InvalidAmount();

    (uint256 balanceABefore, uint256 balanceBBefore) = _getBalances();
    uint256 currentTotalLiquidity = lpToken.totalSupply();

    if (currentTotalLiquidity == 0) {
        if (amountADesired < amountAMin) revert InsufficientAAmount();
        if (amountBDesired < amountBMin) revert InsufficientBAmount();

        amountA = amountADesired;
        amountB = amountBDesired;
    } else {
        uint256 amountBOptimal = _quote(
            amountADesired,
            balanceABefore,
            balanceBBefore
        );

        if (amountBOptimal <= amountBDesired) {
            if (amountADesired < amountAMin) revert InsufficientAAmount();
            if (amountBOptimal < amountBMin) revert InsufficientBAmount();

            amountA = amountADesired;
            amountB = amountBOptimal;
        } else {
```

Slika 3: Izsek kode pametne pogodbe za likvidnostni bazen z logiko za dodajanje likvidnosti

(Vir: Lastni vir)

Na sliki 4 je prikazano nadaljevanje izseka kode pametne pogodbe za likvidnostni bazen, kjer se po prenosu sredstev preveri dejansko prejet znesek in izračuna količino novo izdanih LP-žetonov.

```
            amountB = amountBOptimal;
        } else {
            uint256 amountAOptimal = _quote(
                amountBDesired,
                balanceBBefore,
                balanceABefore
            );

            if (amountAOptimal < amountAMin) revert InsufficientAAmount();
            if (amountBDesired < amountBMin) revert InsufficientBAmount();

            amountA = amountAOptimal;
            amountB = amountBDesired;
        }
    }

    tokenA.safeTransferFrom(msg.sender, address(this), amountA);
    tokenB.safeTransferFrom(msg.sender, address(this), amountB);

    (uint256 balanceAAfter, uint256 balanceBAfter) = _getBalances();
    uint256 receivedA = balanceAAfter - balanceABefore;
    uint256 receivedB = balanceBAfter - balanceBBefore;

    if (receivedA != amountA || receivedB != amountB) {
        revert UnsupportedTokenBehavior();
    }

    if (currentTotalLiquidity == 0) {
        liquidityMinted = _sqrt(receivedA * receivedB);
    } else {
        uint256 liquidityA = (receivedA * currentTotalLiquidity) /
            balanceABefore;
        uint256 liquidityB = (receivedB * currentTotalLiquidity) /
            balanceBBefore;
```

Slika 4: Nadaljevanje izseka kode pametne pogodbe za likvidnostni bazen

(Vir: Lastni vir)

5.1.3 Pogodba za staking

Staking pogodba omogoča uporabniku, da zaklene osnovne žetone in za to prejema časovno odvisne nagrade. Za vsakega uporabnika se hranijo vloženo stanje, čas zadnje posodobitve in nabrane neizplačane nagrade. Ob vsaki pomembni akciji, torej ob »*stake*«, »*unstake*« ali »*claimRewards*«, se najprej izvede posodobitev nagrad, s katero se pravilno preračuna pripadajoči znesek. Pogodba dodatno preprečuje čezmerno izplačilo, tako da je izplačilo omejeno na dejansko predhodno financirani *reward pool*.

V teoretičnem delu se ta implementacija navezuje predvsem na poglavje 2.3.3, kjer je opisan *staking* mehanizem in kako deluje, ter na poglavje 4.4.4, kjer je bil definiran uporabniški primer

stakinga. Praktični del je ta koncept prenesel v ločeno pogodbo, ki omogoča pregledno vodenje evidence, jasno logiko nagrajevanja in večjo modularnost sistema.

Z varnostnega vidika se ta pogodba navezuje na teoretična poglavja 3.2, 3.4 in 3.5, kjer so poudarjeni preverjanje vhodov, zaščita pred napačnimi stanji in kontrola dostopa. Pri implementaciji so bili zato uporabljeni pogoji za preverjanje dovoljenih operacij, mehanizmi za preprečevanje ponovnega zahtevka iste nagrade ter jasna pravila za obravnavo dviga vloženih sredstev. Posebna pozornost je bila namenjena tudi logiki nagrajevanja, da ostane pregledna in preverljiva, saj je zaupanje v pravilnost *staking* sistema povezano s preglednostjo delovanja pogodbe.

Na sliki 5 je prikazan izsek kode pametne pogodbe za *staking*, ki vsebuje osnovno logiko za vlaganje in dvig žetonov ter sprotno posodabljanje uporabniških nagrad.

```
function stake(uint256 amount) external nonReentrant {
    if (amount == 0) revert InvalidAmount();

    _updateRewards(msg.sender);

    stakingToken.safeTransferFrom(msg.sender, address(this), amount);

    stakedBalance[msg.sender] += amount;
    totalStaked += amount;

    emit Staked(msg.sender, amount);
}

function unstake(uint256 amount) external nonReentrant {
    if (amount == 0) revert InvalidAmount();
    if (stakedBalance[msg.sender] < amount)
        revert InsufficientStakedBalance();

    _updateRewards(msg.sender);

    stakedBalance[msg.sender] -= amount;
    totalStaked -= amount;

    stakingToken.safeTransfer(msg.sender, amount);

    emit Unstaked(msg.sender, amount);
}
```

Slika 5: Izsek kode pametne pogodbe za *staking*

(Vir: Lastni vir)

Na sliki 6 je prikazan izsek kode pametne pogodbe za *staking*, ki vsebuje logiko za zahtevek in izplačilo nagrad uporabniku.

```
    function claimRewards() external nonReentrant {
        _updateRewards(msg.sender);

        uint256 reward = pendingRewards[msg.sender];
        if (reward == 0) revert NoRewardsToClaim();

        pendingRewards[msg.sender] = 0;
        reservedRewards -= reward;

        stakingToken.safeTransfer(msg.sender, reward);

        emit RewardsClaimed(msg.sender, reward);
    }
}
```

Slika 6: Izsek kode pametne pogodbe za *staking* z logiko za *claim* nagrad

(Vir: Lastni vir)

5.1.4 Pogodba za yield farming

Yield farming pogodba je zasnovana podobno kot *staking* pogodba, vendar kot vhodno sredstvo ne uporablja osnovnega ERC-20 žetona, temveč LP-žeton likvidnostnega bazena. Uporabnik tako najprej doda likvidnost v DEX, prejme LP-žeton, nato pa ta LP-žeton vloži v *farming* pogodbo. V zameno pridobiva dodatne nagrade v ločenem *reward* žetonu. Takšen pristop jasno loči navadni *staking* od *farming* mehanizma in omogoča bolj realističen DeFi prototip, v katerem se nagrade ne izplačujejo iz istega sredstva, ki je bilo vloženo.

Ta modul se neposredno navezuje na teoretično poglavje 2.3.3, kjer je razložen koncept *yield farminga*, ter na poglavje 4.4.5, kjer je bil definiran uporabniški primer *farming* funkcionalnosti. V praktičnem delu je bil ta koncept prenesen v pogodbo, ki združuje logiko LP-vložkov in dodatnega nagrajevanja, s čimer se povezujejo funkcionalnosti likvidnostnega bazena, *stakinga* in ekonomskega modela aplikacije.

Z varnostnega vidika se pogodba navezuje na teoretična poglavja 3.2 in 3.5, predvsem v delu, ki obravnava preverjanje pogojev, obrambo pred zlorabami in uporabno preverjenih razvojnih vzorcev. Pri implementaciji so bili zato uporabljeni mehanizmi za validacijo vložkov, nadzor

nad izplačilom nagrad in omejevanje nedovoljenih prehodov med stanji pogodbe. Takšen pristop zmanjšuje možnost logičnih napak in prispeva k zanesljivejšemu delovanju *farming* modula.

Na sliki 7 je prikazan izsek kode pametne pogodbe za *yield farming*, ki vsebuje logiko za vlaganje LP-žetonov, njihov dvig ter zahtevek za nagrade.

```
function stake(uint256 amount) external nonReentrant {
    if (amount == 0) revert InvalidAmount();

    _updateRewards(msg.sender);

    lpToken.safeTransferFrom(msg.sender, address(this), amount);

    stakedBalance[msg.sender] += amount;
    totalStaked += amount;

    emit Staked(msg.sender, amount);
}

function unstake(uint256 amount) external nonReentrant {
    if (amount == 0) revert InvalidAmount();
    if (stakedBalance[msg.sender] < amount)
        revert InsufficientStakedBalance();

    _updateRewards(msg.sender);

    stakedBalance[msg.sender] -= amount;
    totalStaked -= amount;

    lpToken.safeTransfer(msg.sender, amount);

    emit Unstaked(msg.sender, amount);
}

function claimRewards() external nonReentrant {
    _updateRewards(msg.sender);

    uint256 reward = pendingRewards[msg.sender];
    if (reward == 0) revert NoRewardsToClaim();

    pendingRewards[msg.sender] = 0;
```

Slika 7: Izsek kode pametne pogodbe za *yield farming*

(Vir: Lastni vir)

5.1.5 Varnostni mehanizmi pri implementaciji pametnih pogodb

Varnostni mehanizmi pri implementaciji pametnih pogodb niso bili obravnavani kot dodatna izboljšava aplikacija, temveč kot temeljni del razvoja sistema. V teoretičnem delu je bilo večkrat poudarjeno, da DeFi aplikacije zaradi neposrednega upravljanja digitalnih sredstev zahtevajo visoko stopnjo varnostne discipline, zato je bilo treba ta načela dosledno prenesti tudi v praktični del implementacije.

Ta podtočka se neposredno navezuje na poglavja 3.2, 3.4 in 3.5, kjer so obravnavane najpogostejše ranljivosti pametnih pogodb, varnost komunikacije med sloji sistema ter priporočene razvojne prakse. Pri implementaciji so bili uporabljeni varnostni vzorci, kot so preverjanje vhodnih parametrov, omejevanje dostopa do administrativnih funkcij, uporaba preverjenih knjižnic ter zaščita pred tipičnimi napakami, ki bi lahko povzročile nepravilno izvajanje logike ali nepooblaščen dostop do sredstev.

Posebno vlogo ima tudi razvojni vzorec *check-effect-interaction*, ki je bil omenjen že v poglavju 2.4.1. Ta pristop določa, da pogodba najprej preveri pogoje, nato posodobi stanje in šele zatem izvede zunanjo interakcijo. V praktični implementaciji je bil ta vzorec pomemben predvsem pri funkcijah, ki vključujejo prenos sredstev ali interakcijo z drugimi pogodbami, saj zmanjšuje tveganje napadov s ponovnim vstopom. Prav tako je bila uporabljena tudi zaščita pred ponovnim vstopom na osnovi *mutex* logike, s katero se prepreči večkratni vstop v kritični del funkcije med njenim izvajanjem.

Poleg tega so bile administrativne funkcije omejene z ustreznimi kontrolami dostopa, medtem ko so bile standardne funkcionalnosti, kot so žetonski prenosi ali osnovne varnostne komponente, naslonjene na preverjene knjižnice. Takšna odločitev je skladna s teoretičnim poudarkom, da se pri razvoju DeFi aplikacij smiselno opreti na preverjene gradnike namesto ročne implementacije vseh osnovnih mehanizmov.

Za boljši pregled implementiranih varnostnih mehanizmov je v tabeli 1 prikazano, kateri varnostni vzorci in zaščitni mehanizmi so bili uporabljeni v posameznih pametnih pogodbah.

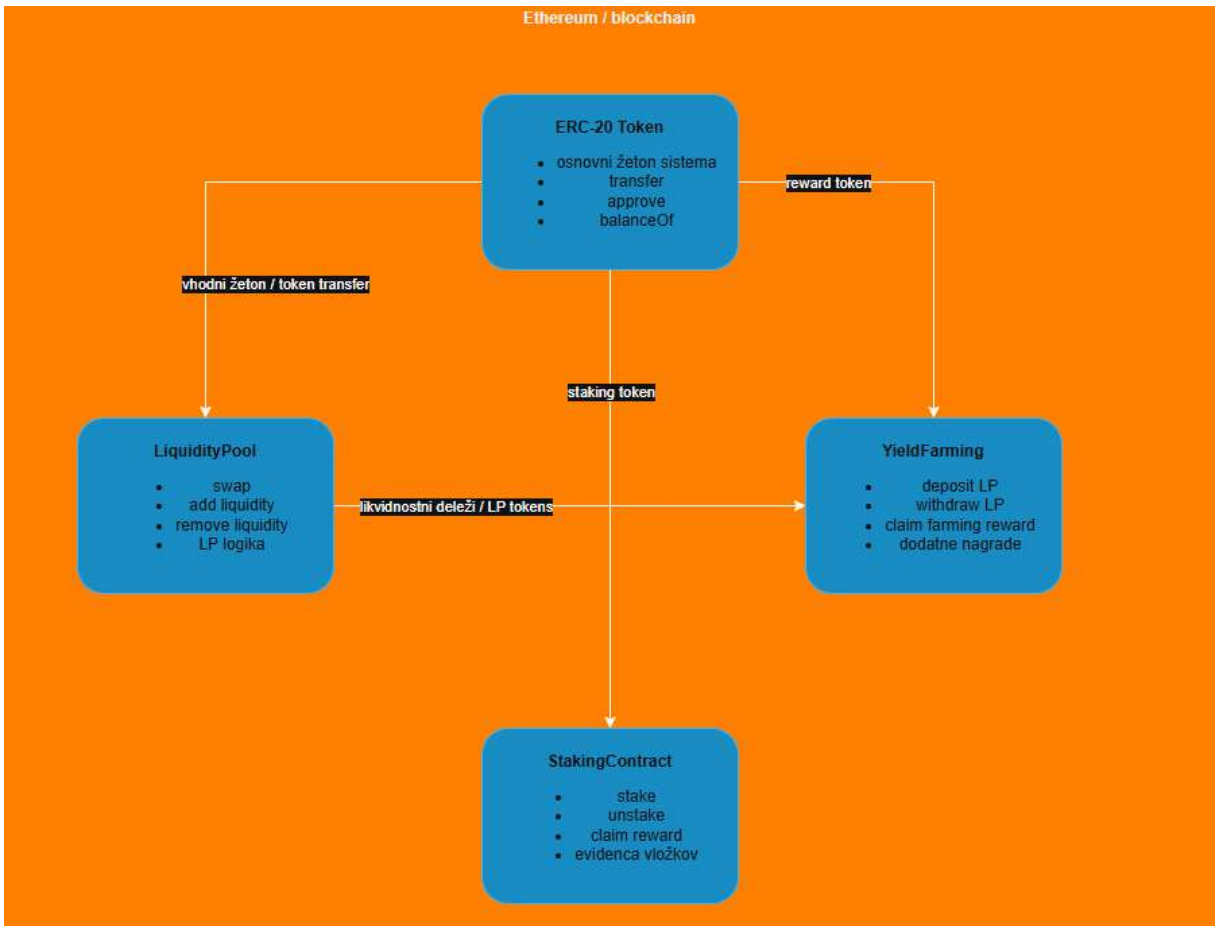
Tabela 1: Varnostni mehanizmi

Pogodba	**Validacija vhodov**	***Check-effect-interaction***	***Mutex*/ *Reentrancy* zaščita**	**Kontrola dostopa**	**Preverjene knjižnice**	**Opomba**
Žetonska pogodba	Da	Delno	Ni potrebno/ osnovno	Da/ omejeno	*OpenZeppelin*	Osnovna ERC-20 logika
Pogodba za likvidnostni bazen	Da	Da	Da	Da	*OpenZeppelin*	*Swap, add/remove liquidity*
Pogodba za *staking*	Da	Da	Da	Da	*OpenZeppelin*	*Stake, unstake, Claim rewards*
Pogodba za *yield farming*	Da	Da	Da	Da	OpenZeppelin	*LP deposit, farming rewards*

(Vir: Lastni vir)

Iz tabele 5.1.1 je razvidno, da so bili varnostni mehanizmi vključeni v vse ključne pogodbe, vendar glede na njihovo funkcijo niso uporabljeni popolnoma enako. Največja stopnja zaščite je bila namenjena pogodbam, ki neposredno upravljajo sredstva uporabnikov in izvajajo spremembe stanja sistema, saj ti deli predstavljajo najpomembnejšo napadalno površino aplikacije.

Modularna zasnova implementiranih pametnih pogodb in njihove medsebojne funkcionalne povezave so prikazane na diagramu na sliki 8.



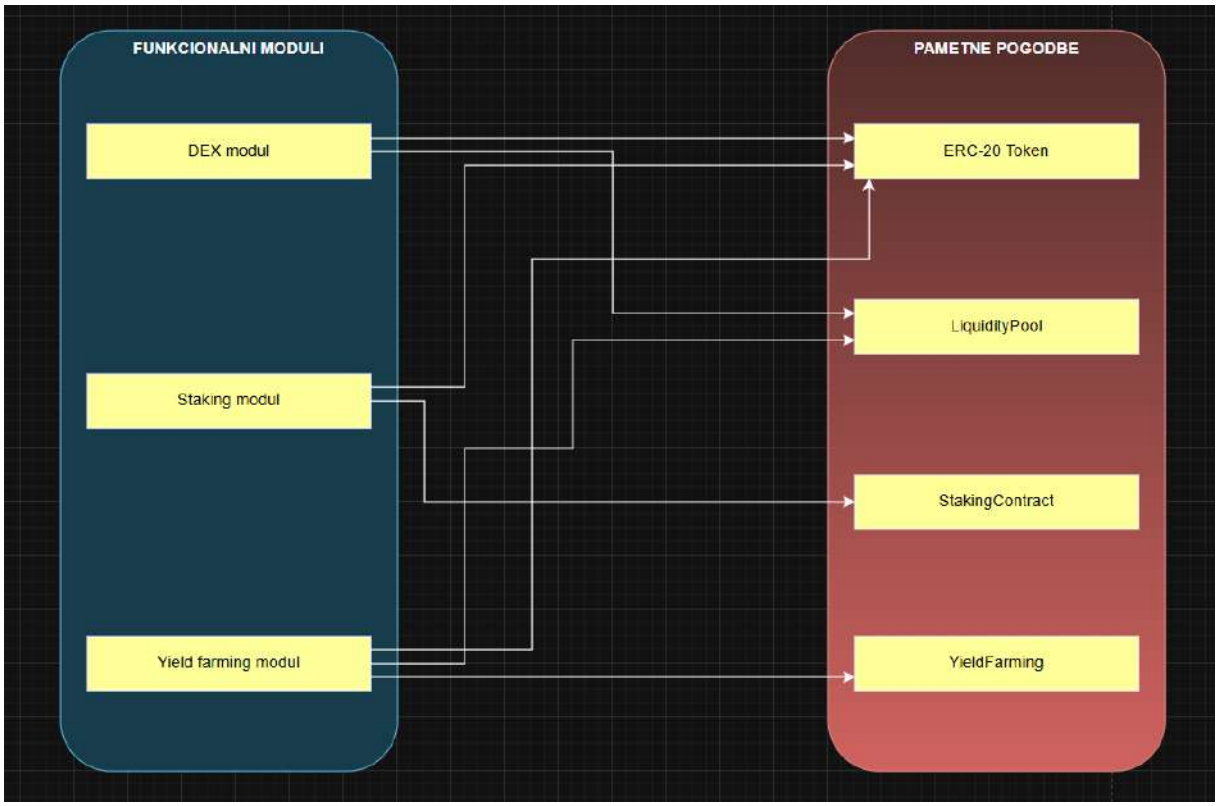
Slika 8: Modularna zasnova pametnih pogodb in njihove funkcionalne povezave

(Vir: Lastni vir)

Diagram na sliki 8 prikazuje, da je poslovna logika aplikacije razdeljena med več ločenih pametnih pogodb, pri čemer vsaka pokriva specifičen funkcionalni del sistema. Takšna zasnova izboljšuje preglednost implementacije, pri čemer olajša testiranje in omogoča jasnejšo razmejitev odgovornosti med posameznimi komponentami.

Povezava med ključnimi funkcionalnostmi aplikacije in implementiranimi pametnimi pogodbami je prikaza z diagramom na sliki 9.

Diagram prikazuje, da posamezne funkcionalnosti aplikacije niso vezane zgolj na eno pogodbo,



Slika 9: Povezava funkcionalnosti aplikacije in pametnih pogodb

(Vir: Lastni vir)

temveč pogosto vključujejo sodelovanje več modulov hkrati. Takšna zasnova je značilna za DeFi sisteme, kjer so poslovni procesi porazdeljeni med več medsebojno povezanim pametnih pogodb.

5.2 Izdelava spletnega uporabniškega vmesnika (frontend)

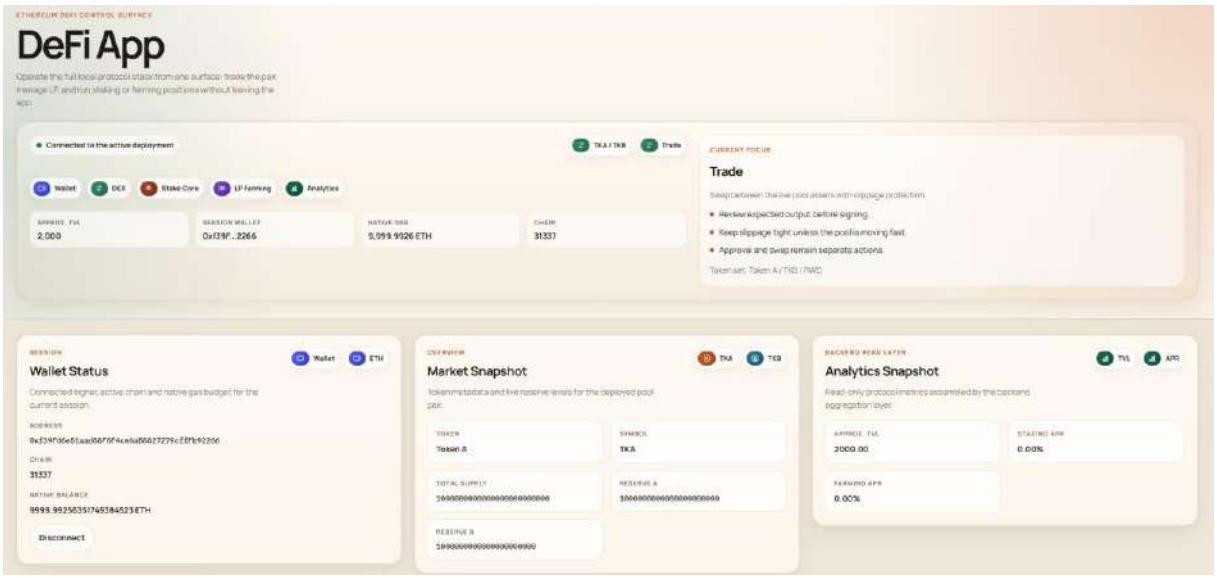
Frontend del aplikacije je izdelan v okolju React in TypeScript ter je razdeljen na več modularnih komponent. Glavni zaslon aplikacije je sestavljen iz nadzorne plošče in delovnega prostora z zavihki. V mapah *components* in *hooks* so ločene vizualne komponente ter Web3 logika za posamezne funkcionalne sklope, kot so *swap*, dodajanje likvidnosti, *staking* in *yield*

farming. Takšen pristop omogoča boljšo preglednost, lažje testiranje in preprostejša nadaljnja razširjanja aplikacije.

Ta del razvoja se neposredno navezuje na teoretični del poglavij 2.4.3 in 4.2, kjer je bilo poudarjeno, da mora biti uporabniški vmesnik v DeFi aplikaciji zasnovan pregledno, modularno in varno. V skladu s tem je bil *frontend* razdeljen na več funkcionalnih modulov, pri čemer vsak pokriva svoje področje delovanja. Med ključne module sodijo povezava s kripto denarnico, modul DEX, modul za dodajanje in odstranjevanje likvidnosti, *staking* modul, *yield farming* modul ter nadzorna plošča za pregled uporabniških podatkov kakor tudi portfelj za pregled uporabniških sredstev.

Uporabniški vmesnik je hkrati povezan z ostalimi sloji sistema. Na eni strani komunicira z denarnico in pametnimi pogodbami, na drugi strani pa po potrebi pridobiva dodatne informativne podatke iz podpornega *backend* sloja. Takšna zasnova omogoča, da decentralizirano jedro ostaja nosilec poslovne logike, *frontend* pa uporabniku zagotavlja pregledno, razumljivo in varno uporabo aplikacije.

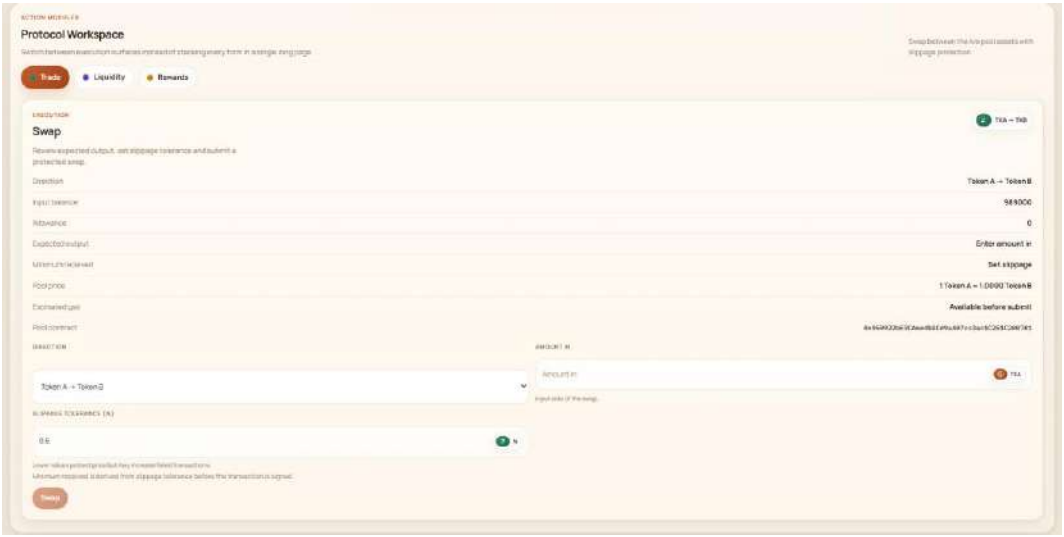
Na sliki 10 je prikazan začetni zaslon aplikacije z nadzorno ploščo, pregledom stanja denarnice, tržnim pregledom in *backend* analitičnim slojem.



Slika 10: Začetni zaslon aplikacije z nadzorno ploščo in pregledom portfelja

(Vir: Lastni vir)

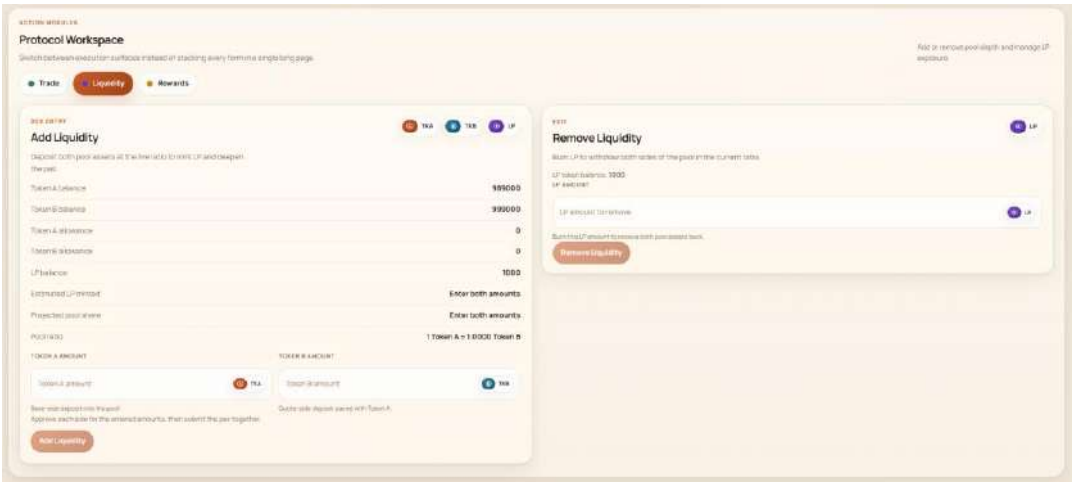
Na sliki 11 je prikazan modul DEX za zamenjavo žetonov, kjer uporabnik vnese vhodni znesek, nastavi toleranco zdrsa cene in pregleda parametre transakcije.



Slika 11: Modul DEX za zamenjavo žetonov

(Vir: Lastni vir)

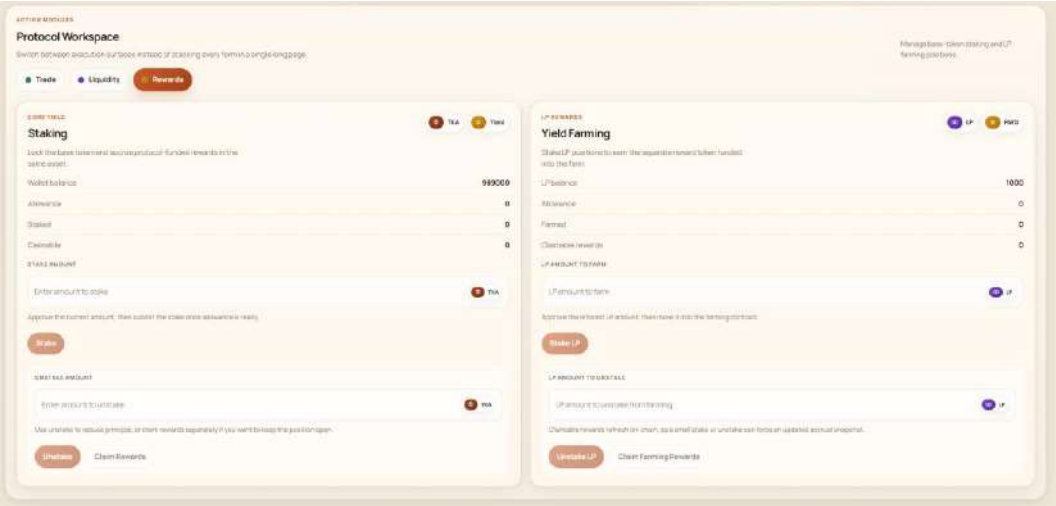
Na sliki 12 je prikazan modul za upravljanje likvidnosti, ki vključuje dodajanje in odstranjevanje likvidnosti ter prikaz osnovnih podatkov o LP-poziciji.



Slika 12: Modul za dodajanje in odstranjevanje likvidnosti

(Vir: Lastni vir)

Na sliki 13 je prikazan modul za *staking* in *yield farming*, kjer lahko uporabnik vlaga žetone oziroma LP-žetone ter upravlja nagrade.



Slika 13: Modula za *staking* in *yield farming*

(Vir: Lastni vir)

Na sliki 14 je prikazan primer pri visoki toleranci zdrsa cene, s katerim aplikacija uporabnika opozori na povečano tveganje manj ugodne izvedbe menjave.



Slika 14: Opozorilo pri visoki toleranci zdrsa cene

(Vir: Lastni vir)

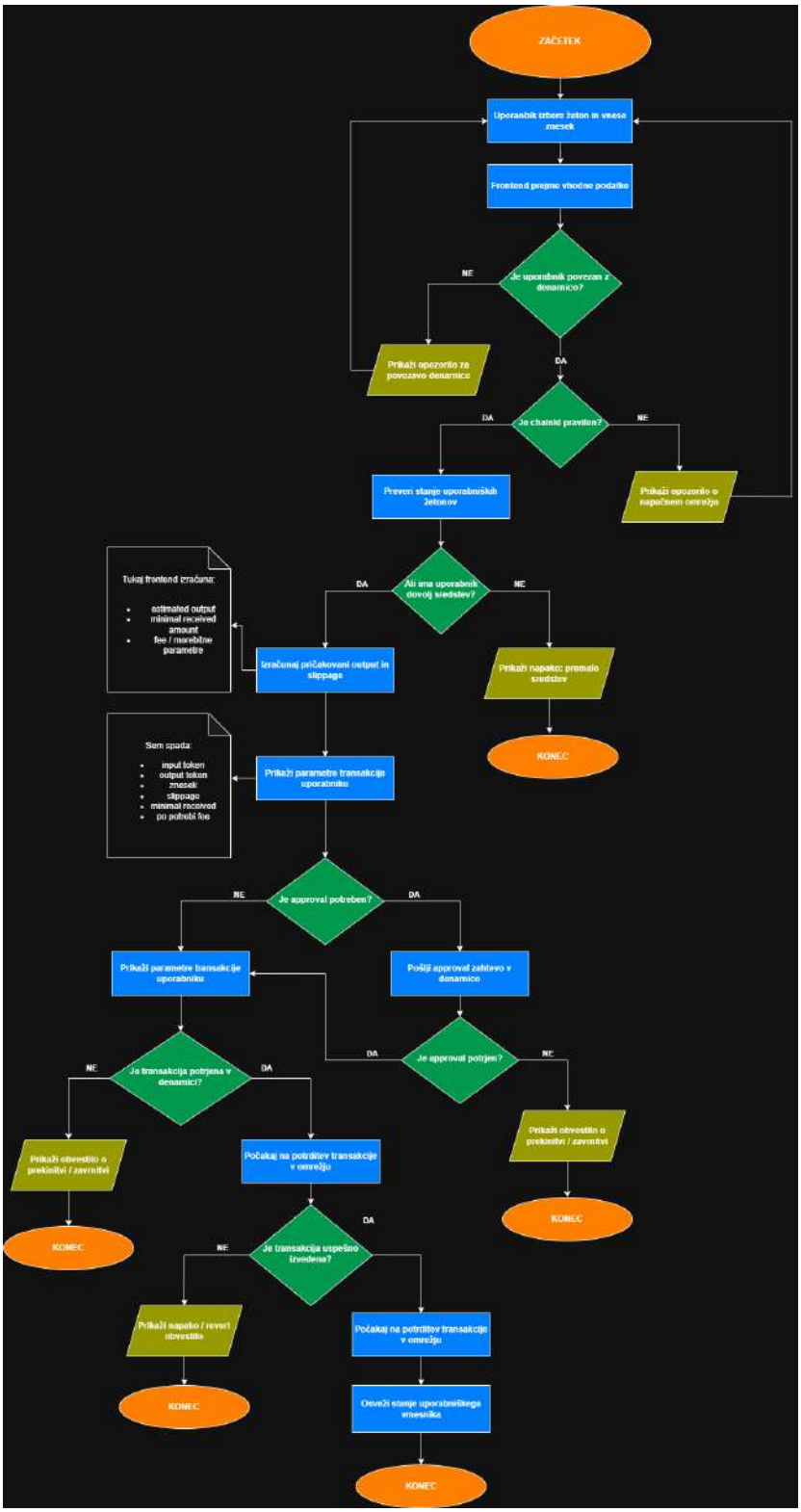
5.3 Razvoj povezave frontend – blockchain (Web3 integracija)

Frontend ne vsebuje trdo kodiranih naslovov pogodb, temveč jih pridobi iz okolijskih spremenljivk. V datoteki »*contracts.ts*« so definirani zahtevani naslovi za osnovni žeton, drugi žeton, likvidnostni bazen, *staking* in *farming* pogodbo. Če katera od teh vrednosti manjka, aplikacija ob zagonu vrne napako, kar zmanjša verjetnost napačne konfiguracije *frontenda*.

Povezava med *frontendom*, denarnico in pametnimi pogodbami je izvedena z uporabo sklada »*wagmi* + *viem*«. Datoteka »*wagmi.ts*« pripravi konfiguracijo verige, RPC transport in injicirani *wallet connector*, medtem ko *hook* »*useWallet*« skrbi za informacije o povezanem naslovu, aktivnem *chainId*, stanju povezave in možnosti preklopa omrežja. Posamezni funkcionalni moduli nato uporabljajo *hooke* »*useReadContract*«, »*useWriteContract*« in »*useWaitForTransactionReceipt*«, s katerimi berejo stanje iz pogodb, oddajo transakcijo in počakajo na njeno potrditev.

Pri interakciji z ERC-20 žetonoma mora uporabnik pred samo akcijo najprej odobriti porabo sredstev izbrane pogodbe. Zaradi tega je v uporabniškem vmesniku jasno ločen korak »*Approve*« od dejanske akcije, na primer »*Swap*«, »*Stake*« ali »*Add Liquidity*«. Tak pristop zmanjša zmedo, uporabniku jasno prikaže, kaj podpisuje v denarnici, in sledi dejanski logiki ERC-20 standarda. Enak vzorec se uporablja v modulu DEX, *staking* modulu in *yield farming* modulu.

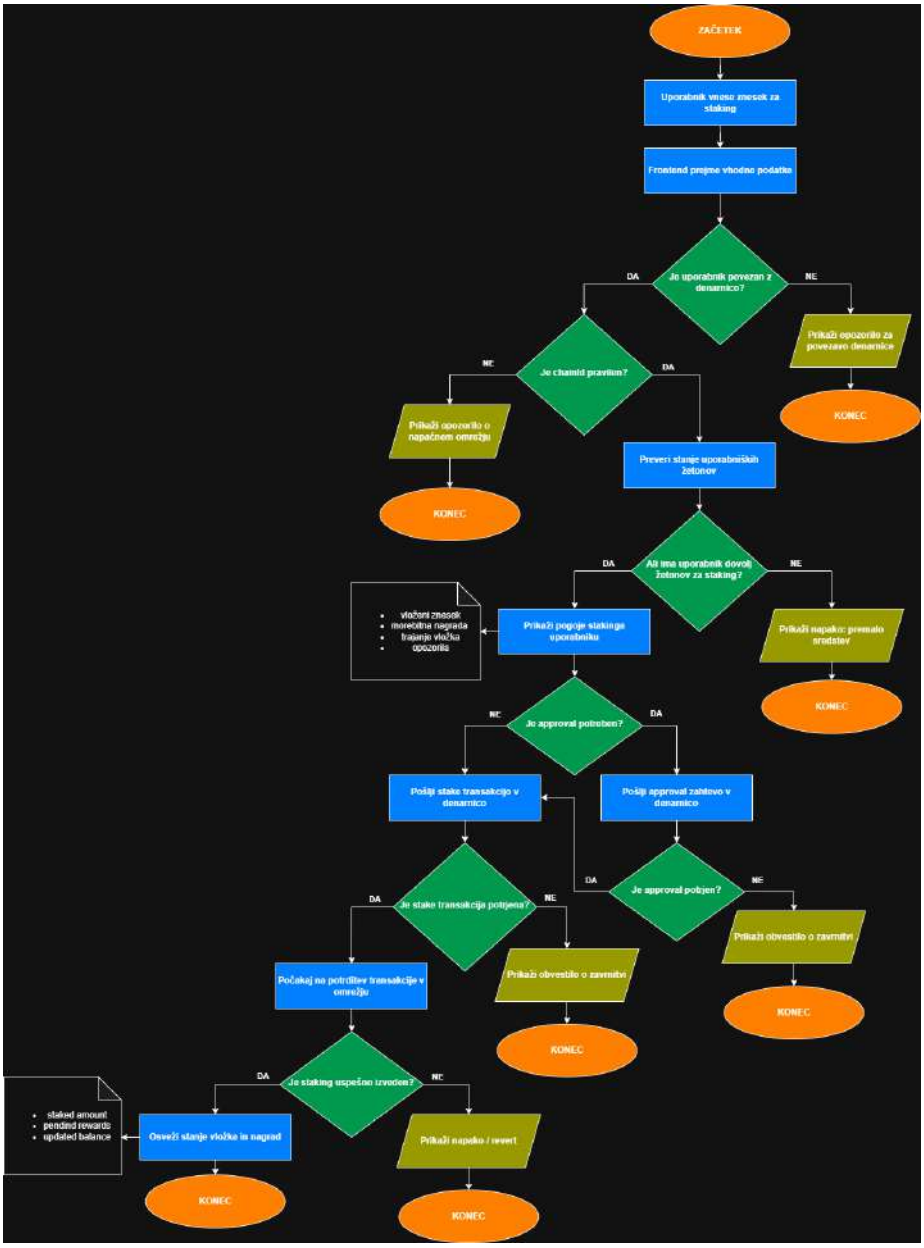
V diagramu poteka na sliki 15 pa je prikazano zaporedje korakov pri izvedbi izmenjave žetonov – od uporabniškega vnosa do potrditve transakcije in osvežitve podatkov v uporabniškem vmesniku. Posebej je razvidna ločitev med fazo odobritve porabe žetonov in dejansko izvedbo *swap* transakcije, kar predstavlja pomemben del varne in pregledne uporabniške interakcije.



Slika 15: Diagram poteka, izvedba zamenjave žetonov

(Vir: Lastni vir)

V diagramu na sliki 16 je prikazan potek *staking* transakcije od uporabniškega vnosa do potrditve na blockchain omrežju. Tudi v tem primeru je pomembna jasna ločitev med odobritvijo žetonov in dejanskim vložkom, saj takšna zasnova izboljšuje preglednost delovanja sistema in zmanjšuje možnost napačne uporabniške potrditve.



Slika 16: Diagram poteka, *staking* transakcija

(Vir: Lastni vir)

5.4 Backend logika in upravljanje podatkov

Čeprav decentralizirano jedro aplikacije temelji na pametnih pogodbah in blockchain omrežju, je bil v okviru prototipa vključen tudi podporni *backend* sloj. Njegova naloga ni izvajanje poslovne logike ali upravljanje sredstev uporabnikov, temveč priprava in posredovanje dodatnih podatkov, ki izboljšujejo preglednost in uporabniško izkušnjo sistema. Takšna zasnova je skladna s teoretičnim delom, predvsem s poglavjema 2.4.4 in 4.3.2, kjer je *backend* opredeljen kot dopolnilna plast decentralizirane aplikacije.

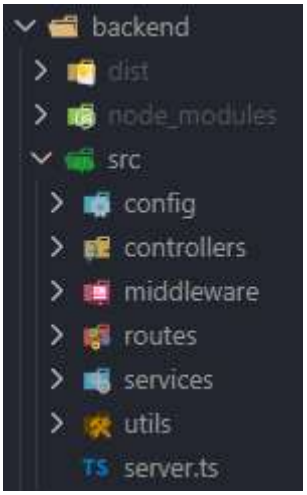
Backend v projektu ni namenjen podpisovanju transakcij ali hranjenju zasebnih ključev, temveč deluje kot podporni *read-only* sloj. Njegova naloga je agregacija podatkov iz verige, priprava preglednih metrik za *frontend* in izračun nekaterih povzetkov, na primer približnega TVL ter APR za *staking* in *farming*. Na ta način blockchain ostane glavni vir resnice za finančno stanje, *backend* pa služi kot pomožni sloj za lažji prikaz podatkov v uporabniškem vmesniku.

Backend je zasnovan kot Node.js in TypeScript aplikacija, zgrajena na osnovi Express strežnika. Pri zagonu prebere konfiguracijo iz okolja, validira naslove pogodb in ustvari »*public client*« za komunikacijo z izbranim Ethereum RPC *endpointom*. Posamezni *endpoint*i nato berejo podatke iz pametnih pogodb in jih vrnejo v preglednejši obliki, prilagojeni potrebam *frontenda*.

Backend trenutno ponuja tri glavne *endpointe*. Prvi je »*GET /api/health*«, ki vrne osnovni podatek o stanju storitve in aktivnem *chainId*. Drugi je »*GET /api/protocol/overview*«, ki *frontendu* pripravi pregled glavnih podatkov protokola, kot so rezerve bazena, TVL, APR pri *stakingu* in APR pri *yield farming* modulu. Tretji je »*GET /api/account/:address/positions*«, ki za izbran naslov vrne stanje osnovnih žetonov, *reward* žetona, LP-žetona ter podatke o *staking* in *farming* poziciji uporabnika.

Z varnostnega vidika je bilo pomembno, da *backend* ostane omejen na *read-only* oziroma podporne funkcije. To pomeni, da ne izvaja občutljivih operacij nad sredstvi uporabnikov, temveč deluje kot pomožni sloj med blockchain omrežjem in uporabniškim vmesnikom. Poleg tega so bile pri njegovi zasnovi upoštevane osnovne prakse, kot so ločevanje okolij, beleženje dostopov in omejevanje nepotrebnih funkcionalnosti, s čimer se zmanjšuje dodatna napadalna površina sistema.

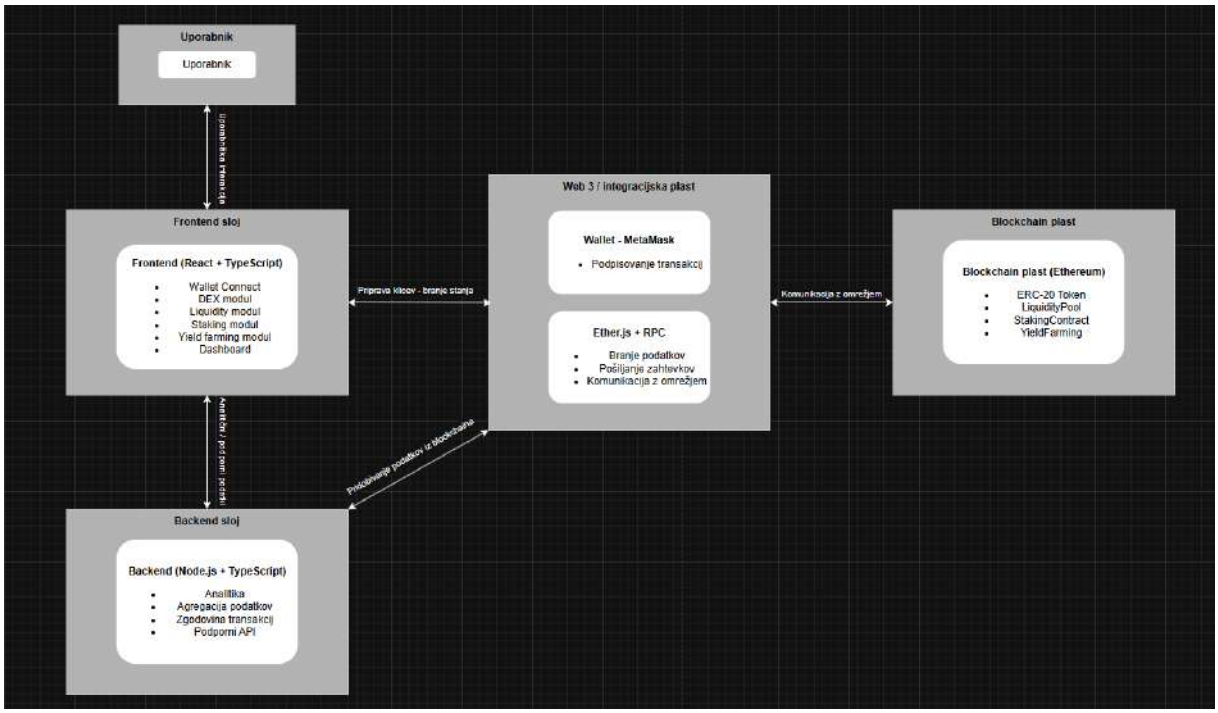
Na sliki 17 je prikazana organizacija *backend* dela aplikacije, ki je zasnovan modularno in vključuje ločene mape za konfiguracije, usmerjanje zahtevkov, storitveni sloj, pomožne funkcije ter osrednjo strežniško datoteko.



Slika 17: Struktura *backend* projekta

(Vir: Lastni vir)

Celotna arhitektura razvite aplikacije je prikazana z diagramom na sliki 18, kjer so predstavljeni glavni sloji sistema in njihove medsebojne povezave.



Slika 18: Arhitekturni diagram celotne aplikacije

(Vir: Lastni vir)

Diagram prikazuje večplastno zasnovo aplikacije, v kateri decentralizirano jedro predstavljajo pametne pogodbe na blockchain omrežju, uporabniški vmesnik pa omogoča interakcijo z njimi prek Web3 integracijske plasti. *Backend* je v arhitekturo vključen kot podporni sloj za pripravo dodatnih podatkov in ne prevzema vloge osrednje točke zaupanja, kar je skladno z zasnovo decentraliziranega sistema.

5.5 Testno okolje: simulacija blockchaina (Hardhat/Ganache)

Pri razvoju DeFi aplikacije je testno okolje predstavljalo pomemben del praktične izvedbe, saj pametnih pogodb po objavi na blockchainu praviloma ni mogoče preprosto spreminjati brez dodatnih posegov v sistem. Zaradi tega je bilo treba ključne funkcionalnosti aplikacije najprej preverjati v nadzorovanem razvojnem okolju, ki omogoča simulacijo blockchaina, nameščanje pogodb, izvajanje transakcij ter preverjanje pravilnosti delovanja posameznih komponent.

Ta del razvoja se neposredno navezuje na teoretično poglavje 2.4.5, kjer so bila kot možna razvojna okolja za simulacijo blockchaina omenjena orodja Hardhat in Ganache. V fazi načrtovanja sta bili obe rešitvi upoštevani kot primerne možnosti, v praktični izvedbi pa je bil

uporabljen Hardhat, saj omogoča celovitejšo podporo razvoju, nameščanju in testiranju pametnih pogodb ter boljši nadzor nad lokalnim razvojnim okoljem.

Postopek *deploymenta* v testnem okolju temelji na Hardhat skripti »*deploy-local-stack.ts*«. Skripta najprej ustvari osnovni žeton A, žeton B in *reward* žeton, nato se razporedi pogodba za likvidnostni bazen, pogodba za *staking* in pogodba za *yield farming*. Po razporeditvi skripta se inicialno napolni bazen z likvidnostjo, financira *reward pool* za *staking* ter financira *reward pool* za *farming*. Na koncu se v terminal izpišejo naslovi razporejenih pogodb, ki se nato prenesejo v ».env« datoteko in jih uporablja *frontend* ter *backend*.

Za razvoj in testiranje je bilo uporabljeno okolje Hardhat. Konfiguracija v datoteki »*hardhat.config.ts*« določa uporabljeno verzijo prevajalnika Solidity ter omogoča delo tako z lokalnim simuliranim omrežjem kot tudi s testnim omrežjem Sepolia. Med lokalnim razvojem se zažene Hardhat node na *chainId* 31337, ki samodejno pripravi več testnih računov z napolnjenim stanjem ETH. Eden od teh računov se nato uvozi v MetaMask, kar omogoča lokalno preizkušanje vseh funkcionalnosti aplikacije prek istega uporabniškega toka kot na pravem omrežju.

Na sliki 19 je prikazan prvi del *deploy* skripte za lokalno testno okolje, kjer se inicializirajo osnovni parametri ter zaporedno namestijo žetonske in DeFi pametne pogodbe.

Na sliki 20 je prikazan drugi del *deploy* skripte, kjer se izvede začetna priprava protokola, financiranje nagradnih skladov ter izpis naslovov pogodb za povezavo s *frontend* delom aplikacije.



```
import { network } from 'hardhat';

async function main() {
    const { ethers } = await network.connect();

    const initialSupply = ethers.parseUnits('1000000', 18);
    const liquidityAmount = ethers.parseUnits('1000', 18);
    const rewardFunding = ethers.parseUnits('10000', 18);
    const rewardRatePerSecond = ethers.parseUnits('0.001', 18);

    const tokenA = await ethers.deployContract('ERC20Token', [
        'Token A',
        'TKA',
        initialSupply,
    ]);
    await tokenA.waitForDeployment();

    const tokenB = await ethers.deployContract('ERC20Token', [
        'Token B',
        'TKB',
        initialSupply,
    ]);
    await tokenB.waitForDeployment();

    const rewardToken = await ethers.deployContract('ERC20Token', [
        'Reward Token',
        'RWD',
        initialSupply,
    ]);
    await rewardToken.waitForDeployment();

    const pool = await ethers.deployContract('LiquidityPool', [
        await tokenA.getAddress(),
        await tokenB.getAddress(),
    ]);
    await pool.waitForDeployment();
```

Slika 19: Prvi del *deploy* skripte za lokalno testno okolje

(Vir: Lastni vir)

```
    await (await tokenB.approve(await pool.getAddress(), liquidityAmount)).wait(
    await (
        await pool.addLiquidity(
            liquidityAmount,
            liquidityAmount,
            liquidityAmount,
            liquidityAmount,
        )
    ).wait();

    await (
        await tokenA.approve(await staking.getAddress(), rewardFunding)
    ).wait();
    await (await staking.fundRewards(rewardFunding)).wait();

    await (
        await rewardToken.approve(await farming.getAddress(), rewardFunding)
    ).wait();
    await (await farming.fundRewards(rewardFunding)).wait();

    console.log(`VITE_APP_TOKEN_ADDRESS=${await tokenA.getAddress()}`);
    console.log(`VITE_APP_POOL_ADDRESS=${await pool.getAddress()}`);
    console.log(`VITE_APP_STAKING_ADDRESS=${await staking.getAddress()}`);
    console.log(`VITE_APP_FARMING_ADDRESS=${await farming.getAddress()}`);
    console.log(
        `VITE_APP_REWARD_TOKEN_ADDRESS=${await rewardToken.getAddress()}`,
    );
    console.log(`VITE_APP_TOKEN_B_ADDRESS=${await tokenB.getAddress()}`);
}

main().catch((error) => {
    console.error(error);
    process.exitCode = 1;
});
```

Slika 20: Drugi del *deploy* skripte z inicializacijo protokola in izpisom naslovov pogodb

(Vir: Lastni vir)

5.6 Povezovanje vseh komponent v funkcionalno celoto

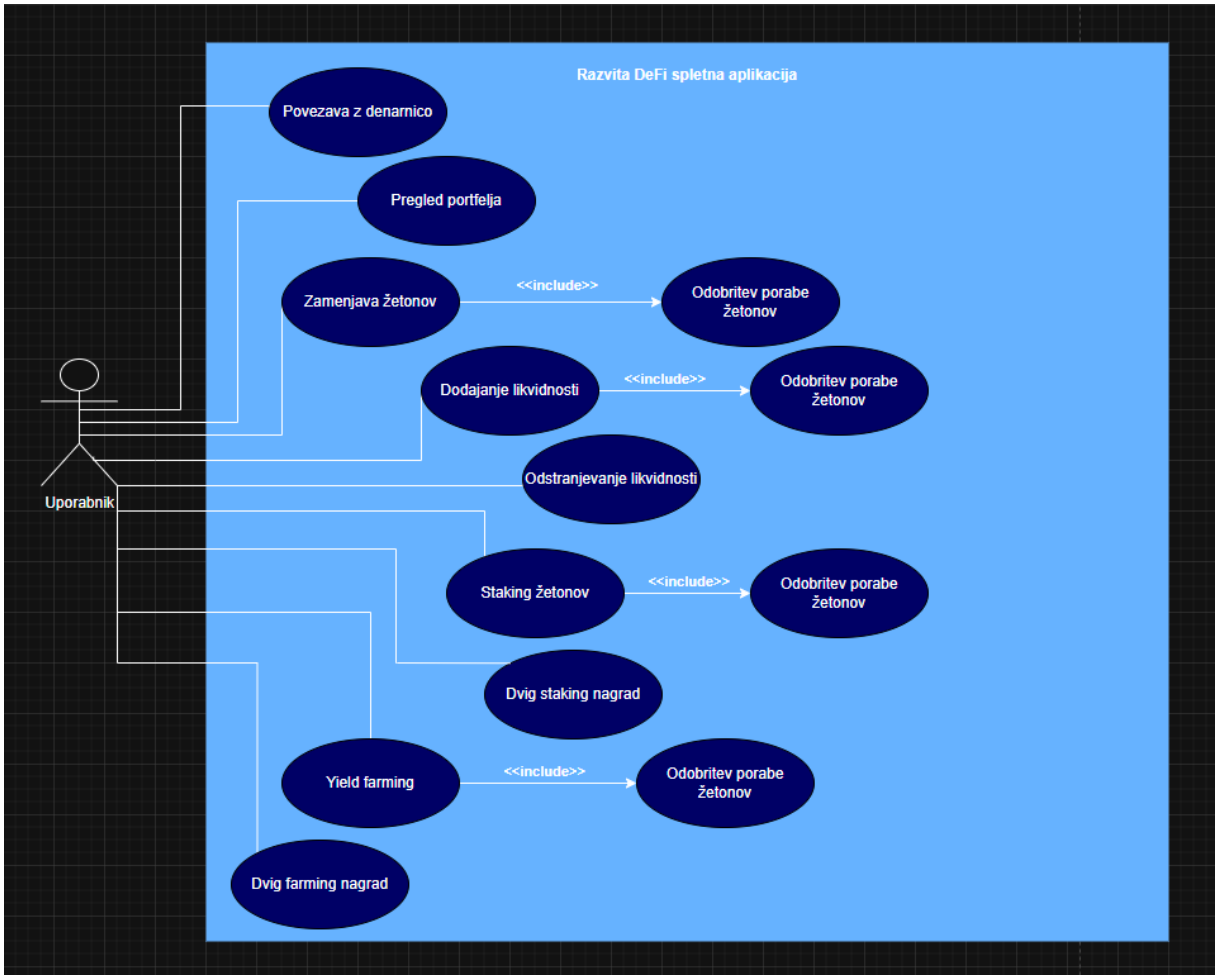
Po implementaciji posameznih delov sistema je sledila faza povezovanja vseh komponent v enotno funkcionalno celoto. V tej fazi je bilo treba zagotoviti, da pametne pogodbe, uporabniški vmesnik, Web3 integracijska plast in podporni *backend* sloj delujejo usklajeno ter da med njimi poteka pravilna izmenjava podatkov. Namen te faze ni bil več razvoj posameznih modulov, temveč preverjanje, ali se vsi deli sistema povezujejo v uporabno DeFi aplikacijo.

Ta del praktičnega razvoja se navezuje predvsem na teoretična poglavja 4.3, 4.4 in 2.4, kjer je bila predstavljena večplastna arhitektura sistema ter vloga posameznih slojev znotraj DeFi aplikacije. V praktični izvedbi je to pomenilo, da je bilo treba po razvoju posameznih komponent povezati naslove pametnih pogodb z uporabniškim vmesnikom, zagotoviti pravilno komunikacijo prek integracijskega sloja ter uskladiti *backend* logiko s podatki, ki jih aplikacija uporablja za dodatne prikaze in analitične funkcije.

Pomemben del te faze je bila tudi priprava ustreznih konfiguracijskih podatkov, kot so naslovi pametnih pogodb, povezava z izbranim omrežjem in uskladitev komunikacije med *frontend* in *backend* delom sistema. Le s pravilno usklajenostjo teh elementov je bilo mogoče doseči, da aplikacija deluje kot enotna rešitev in ne zgolj kot skupek ločenih tehničnih modulov.

Povezovanje vseh komponent je omogočilo preverjanje celotnega uporabniškega toka – od povezave s kripto denarnico in prikaza podatkov do izvedbe transakcije ter osvežitve stanja sistema. S tem je bila zaključena razvojna faza prototipa in vzpostavljena osnova za nadaljnje testiranje, ki je predstavljeno v naslednjem poglavju.

Povezava med ključnimi uporabniškimi funkcionalnostmi razvite aplikacije je prikazana z diagramom primera uporabe na sliki 21, ki povzema scenarije uporabe sistema.



Slika 21: Uporabniški primeri uporabe razvite DeFi aplikacije

(Vir: Lastni vir)

Diagram prikazuje ključne uporabniške funkcionalnosti, ki jih podpira razvita aplikacija. Iz prikaza je razvidno, da sistem združuje več medsebojno povezanih funkcionalnih sklopov, pri

čemer uporabnik prek enotnega vmesnika dostopa do menjalnih, likvidnostnih, *staking* in *farming* funkcionalnosti ter spremlja stanje svojih sredstev.

6 TESTIRANJE IN ANALIZA DELOVANJA

V tem poglavju je predstavljeno testiranje razvite DeFi spletne aplikacije in analiza njenega delovanja. Namen testiranja je preveriti, ali posamezne komponente sistema delujejo pravilno, ali so med seboj ustrezno povezane ter ali aplikacija izpolnjuje funkcionalne in varnostne zahteve, opredeljene v prejšnjih poglavjih.

Testiranje zajema več ravni sistema – od pametnih pogodb in uporabniškega vmesnika do uporabniške izkušnje in varnostnih vidikov rešitve. Na tej osnovi je mogoče oceniti zanesljivost razvite aplikacije in pripraviti podlago za analizo rezultatov ter preverjanje hipotez v zaključnem delu diplomskega dela.

6.1 Testiranje pametnih pogodb (unit test, edge case testiranje)

Testiranje pametnih pogodb je predstavljalo enega najpomembnejših delov preverjanja razvite DeFi aplikacije, saj pametne pogodbe vsebujejo ključno poslovno logiko sistema. Namen testiranja je bil preveriti pravilnost izvajanja osnovnih funkcionalnosti ter odkriti morebitne napake pri obdelavi robnih primerov, ki bi lahko vplivali na stabilnost ali varnost delovanja aplikacije.

Ta del se neposredno navezuje na teoretična poglavja o varnosti pametnih pogodb in testnem okolju, kjer je bilo poudarjeno, da je treba pri DeFi aplikacijah poleg običajnih scenarijev preverjati tudi situacije, v katerih lahko pride do nepravilnih vhodnih podatkov, neustreznih stanj ali poskusov zlorabe logike sistema. V praktičnem delu je bilo zato testiranje usmerjeno tako v običajno uporabo kot tudi v *edge case* scenarije.

Posebej so bile preverjene funkcionalnosti, povezane z zamenjavo žetonov, upravljanjem likvidnosti, *staking* mehanizmom, izračunom nagrad ter *yield farming* logiko. Poleg uspešnih primerov so bili pomembni tudi testi, pri katerih mora sistem transakcijo pravilno zavrniti, na primer pri nezadostnem stanju sredstev, ničelnem vložku ali večkratnem zahtevku za nagrado.

6.1.1 Uporabljen testni pristop in ogrodje

Za unit testiranje pametnih pogodb je bilo uporabljeno okolje Hardhat, ki omogoča lokalno simulacijo Ethereum omrežja in avtomatizirano poganjanje testov. Testi so napisani v jeziku TypeScript, pri čemer je za definicijo testnih scenarijev uporabljeno ogrodje Mocha, za

preverjanje pričakovanih rezultatov pa knjižnica Chai. Interakcije s pogodbami v testih potekajo prek orodij, ki jih omogoča Hardhat skupaj s knjižnico »*ethers.js*«.

Na sliki 22 je prikazan način poganjanja avtomatiziranih *unit* testov v okolju *hardhat*, kot je definiran v skripti projekta.

```
"scripts": {
    "compile": "hardhat compile",
    "test": "hardhat test",
    "clean": "hardhat clean"
},
```

Slika 22: Definiran ukaz za poganjanje *unit* testov v okolju *hardhat*

(Vir: Lastni vir)

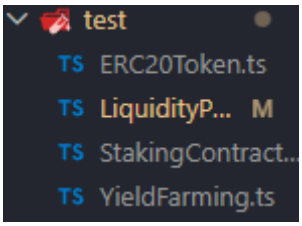
Na sliki 23 je prikazana konfiguracija Hardhat okolja, ki je bila uporabljena za prevajanje pogodb in delo v lokalnem ter testnem omrežju.

```
export default defineConfig({
  plugins: [hardhatToolboxMochaEthersPlugin],
  solidity: {
    profiles: {
      default: {
        version: "0.8.28",
      },
      production: {
        version: "0.8.28",
        settings: {
          optimizer: {
            enabled: true,
            runs: 200,
          },
        },
      },
    },
  },
  networks: {
    hardhatMainnet: {
      type: "edr-simulated",
      chainType: "l1",
    },
    hardhatOp: {
      type: "edr-simulated",
      chainType: "op",
    },
    sepolia: {
      type: "http",
      chainType: "l1",
      url: configVariable("SEPOLIA_RPC_URL"),
      accounts: [configVariable("SEPOLIA_PRIVATE_KEY")]
    },
  },
});
```

Slika 23: Konfiguracija Hardhat razvojnega okolja in testnega okolja

(Vir: Lastni vir)

Na sliki 24 je prikazana organizacija testnih datotek po posameznih pametnih pogodbah, kar omogoča jasno ločitev testnih scenarijev glede na funkcionalni sklop.



Slika 24: Organizacija *unit* testov po posameznih pogodbah

(Vir: Lastni vir)

6.1.2 Testirane funkcionalnosti po posameznih pogodbah

Pri pogodbi »*ERC20Token*« so bile preverjene osnovne lastnosti standarda ERC-20. Testi so zajemali pravilno inicializacijo začetne ponudbe, prenose žetonov med računi ter mehanizem »*approve/allowance/transferFrom*«, ki je ključen za poznejše sodelovanje z DEX, *staking* in *farming* pogodbami.

Pri pogodbi »*LiquidityPool*« je bilo preverjeno dodajanje likvidnosti, izdaja LP-žetonov, odstranitev likvidnosti, pravilno posodabljanje rezerv ter izvedba menjave med obema sredstvoma. Posebej je bilo preverjeno tudi, ali pogodba pravilno uporablja minimalni spremenljivi izhod (»*minAmountOut*«) in ali zavrne transakcije z neveljavnim vhodnim žetonom ali z neskladnimi parametri pri dodajanju likvidnosti.

Pri pogodbi »*StakingContract*« je bilo preverjeno vlaganje osnovnega žetona, dvig vloženih sredstev, izračun in *claim* nagrad ter pravilno delovanje časovno odvisne logike. Poleg normalnih scenarijev je bilo pomembno preveriti tudi ničelne vložke, poskus dviga več sredstev od vloženega zneska, poskus zahtevka za nagrado brez akumuliranih *rewardov* ter omejitev nagrad na dejansko financiran *reward pool*.

Pri pogodbi »*YieldFarming*« je bilo testiranje usmerjeno v delo z LP-žetoni kot vhodnim sredstvom. Preverjeni so bili *staking* LP-žetonov, *unstake*, *claim rewardov*, pravilno delovanje časovne *reward* logike, ter enaki ključni robni primeri kot pri *staking* pogodbi, vključno z omejitvijo izplačila na dejansko financirana *reward* sredstva.

6.1.3 Najpomembnejši edge case scenariji

Pri testiranju niso bili preverjani samo uspešni scenariji, ampak tudi primer, v kateri mora sistem transakcijo varno zavrniti. Med pomembnejšimi robnimi scenariji so bili:

- ničelni zneski pri »*stake*«, »*unstake*« in »*removeLiquidity*«;
- poskus odstranitve več likvidnosti, kot je uporabnik dejansko ima;
- poskus menjave pri previsokem »*minAmountOut*«;
- poskus zahtevka za nagrado brez akumuliranih *rewardov*;
- poskus izplačila *rewardov* nad dejansko financiranim *reward poolom*;
- preverjanje, da se *rewardi* ne akumulirajo za obdobje pred dejanskim financiranjem *reward* sklada.

Na sliki 25 je prikazan primer *edge case* testiranja, pri katerem se preverja pravilna zavrnitev menjave, kadar je zahtevani minimalni izhod previsok glede na stanje v bazenu.

```
    expect(reserveA).to.equal(amountA + swapAmountIn);
    expect(reserveB).to.equal(amountB - expectedAmountOut);
});

it('should revert swap when minAmountOut is too high', async function () {
    const { ethers, tokenA, tokenB, pool } = await deployFixture();

    const amountA = ethers.parseUnits('1000', 18);
    const amountB = ethers.parseUnits('1000', 18);

    await (await tokenA.approve(await pool.getAddress(), amountA)).wait();
    await (await tokenB.approve(await pool.getAddress(), amountB)).wait();
    await (await pool.addLiquidity(amountA, amountB, amountA, amountB)).wait();

    const swapAmountIn = ethers.parseUnits('100', 18);

    await (await tokenA.approve(await pool.getAddress(), swapAmountIn)).wait();

    const unrealisticMinAmountOut = ethers.parseUnits('200', 18);

    await expect(
        pool.swap(
            await tokenA.getAddress(),
            swapAmountIn,
            unrealisticMinAmountOut,
        ),
    ).to.be.revertedWithCustomError(pool, 'InsufficientOutputAmount');
});

it('should revert swap when tokenIn is invalid', async function () {
    const { ethers, tokenA, tokenB, pool } = await deployFixture();
```

Slika 25: Primer *edge case* testa za preverjanje *slippage* zaščite pri zamenjavi sredstev

(Vir: Lastni vir)

Na sliki 26 je prikazan primer robnega scenarija pri *staking* pogodbi, kjer se preverja, da izplačilo nagrad ne preseže dejansko financiranega *reward* sklada.

```
it('should cap rewards to the funded reward pool', async function () {
    const { ethers, owner, token, staking } = await deployFixture();

    const stakeAmount = ethers.parseUnits('1000', 18);
    const rewardFunding = ethers.parseUnits('10', 18);

    await (await token.approve(await staking.getAddress(), rewardFunding)).wait();
    await (await staking.fundRewards(rewardFunding)).wait();

    await (await token.approve(await staking.getAddress(), stakeAmount)).wait();
    await (await staking.stake(stakeAmount)).wait();

    await ethers.provider.send('evm_increaseTime', [100]);
    await ethers.provider.send('evm_mine', []);

    const ownerBalanceBefore = await token.balanceOf(owner.address);

    await (await staking.claimRewards()).wait();

    const claimedReward = (await token.balanceOf(owner.address)) - ownerBalanceBefore;

    expect(claimedReward).to.equal(rewardFunding);
    expect(await staking.pendingRewards(owner.address)).to.equal(0n);
    expect(await staking.reservedRewards()).to.equal(0n);
});
```

Slika 26: Primer *edge case* testa za omejitev izplačila nagrad na financirani *reward pool*

(Vir: Lastni vir)

6.1.4 Rezultati unit testiranja

Ob končnem zagonu testov je bilo uspešno izvedenih 29 *unit* testov. Testi so bili razporejeni med štiri glavne pametne pogodbe, pri čemer je največji testni nabor pripadal pogodbi za likvidnostni bazen, saj vsebuje največ različnih funkcionalnosti in robnih primerov.

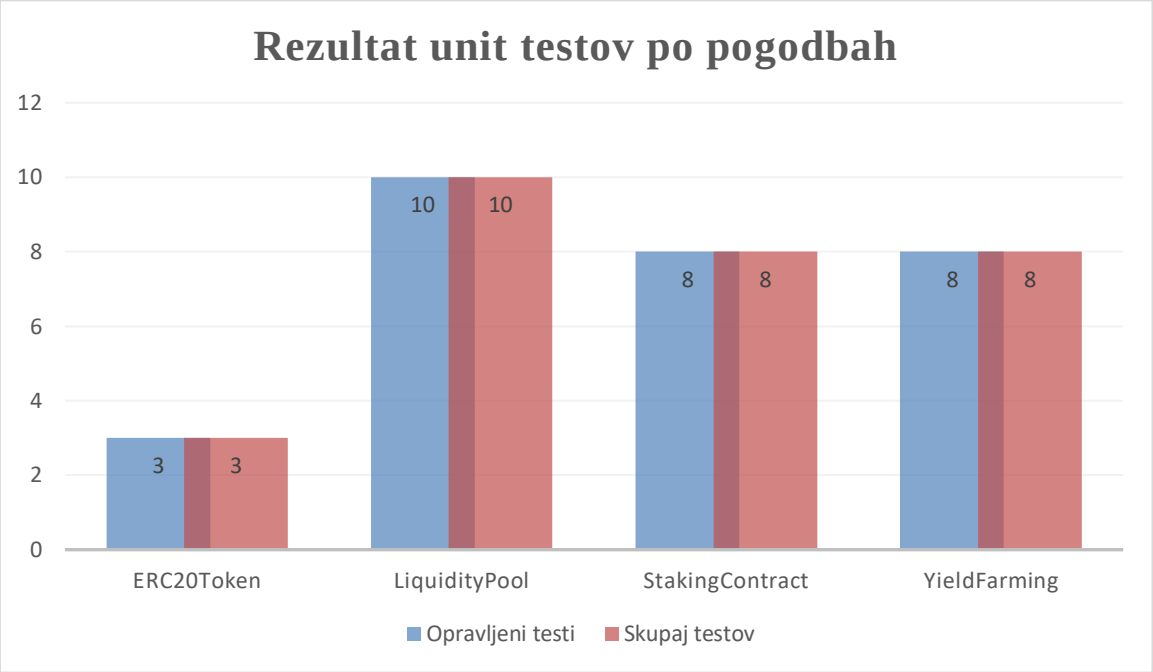
Pregled števila testov, preverjanih funkcionalnosti in rezultatov *unit* testiranja po posameznih pogodbah je prikazan v tabeli 2.

Tabela 2: Pregled *unit* testov po pametnih pogodbah

Pogodba	Število testov	Ključne funkcionalnosti	Rezultat
ERC20Token	3	Inicialna pogodba, *transfer, approve / allowance / transferFrom*	3 / 3
LiquidityPool	10	*Add liquidity, remove liquidity, LP minting, swap, minAmountOut*, validacija tokenov	10 / 10
StakingContract	8	*Stake, unstake, claim rewards, funding reward pool, reward cap*	8 / 8
YieldFarming	8	*LP stake, LP unstake, claim rewards, funding reward pool, reward cap*	8 / 8
Skupaj	29	Celoten *smart contract* sloj prototipa	29 / 29

(Vir: Lastni vir)

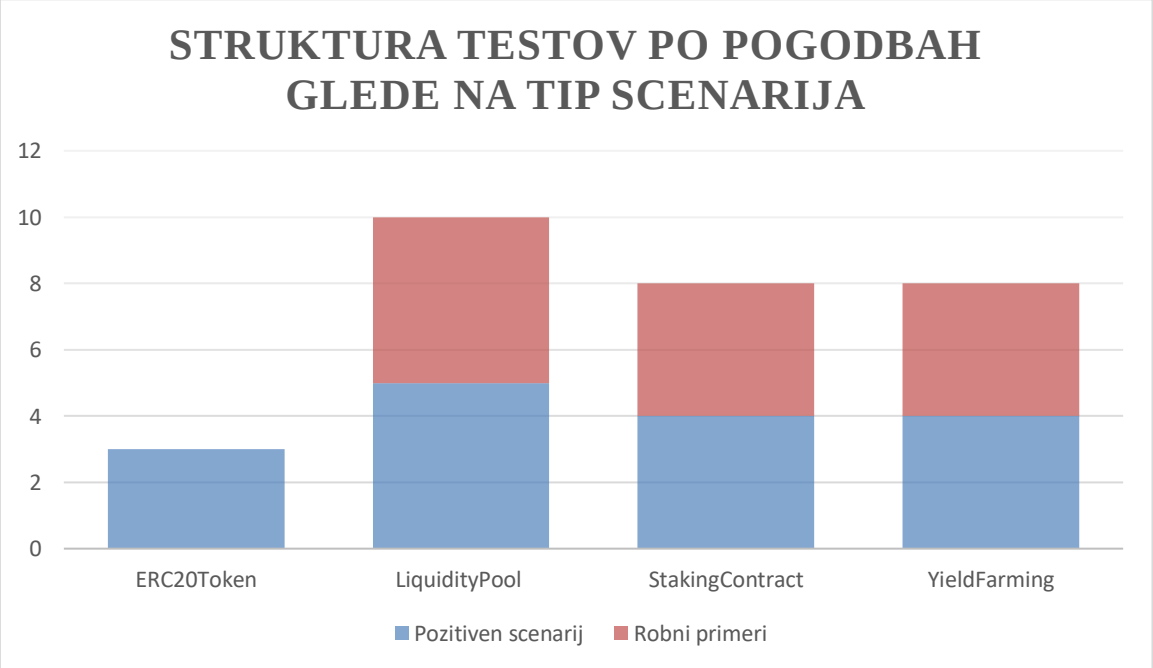
Na stolpničnem grafikonu 1 so prikazani rezultati *unit* testov po pogodbah.



Grafikon 1: Rezultat *unit* testov po pogodbah

(Vir: Lastni vir)

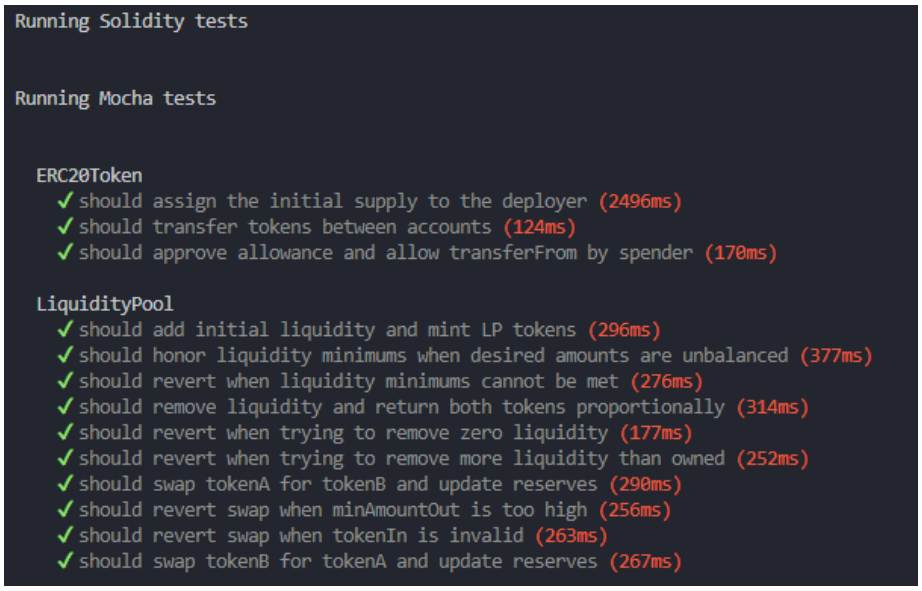
Na stolpničnem grafikonu 2 je prikazana struktura testov po pogodbah glede na tip scenarija.



Grafikon 2: Struktura testov po pogodbah glede na tip scenarija

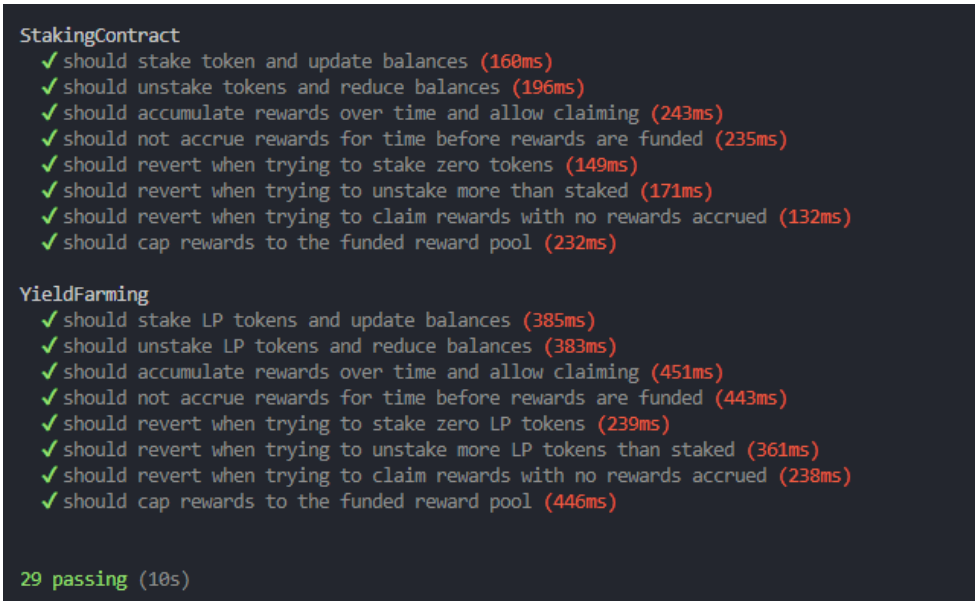
(Vir: Lastni vir)

Na sliki 27 in sliki 28 je prikaz rezultata v terminalu za uspešno izvedene teste.



Slika 27: Rezultat za uspešno izvedene teste 1

(Vir: Lastni vir)



Slika 28: Rezultat za uspešno izvedene teste 2

(Vir: Lastni vir)

6.2 Testiranje frontend funkcionalnosti

Testiranje *frontend* funkcionalnosti je bilo izvedeno z združenim pristopom avtomatiziranega in ročnega preverjanja. Avtomatizirani del testiranja je bil pripravljen v okolju React in TypeScript z uporabo ogrodja Vitest ter knjižnice @testing-library/react, kar je omogočilo preverjanje pravilnega izrisa uporabniškega vmesnika. Pri tem so bili avtomatizirano preverjeni

predvsem začetno stanje aplikacije brez povezane denarnice, stanje ob napačnem omrežju, prikaz glavne nadzorne plošče ter logika ločevanja med *approval* in *action* koraki v modulih za zamenjavo žetonov in dodajanje likvidnosti.

Ta del se navezuje na teoretična poglavja o uporabniškem vmesniku, Web3 integraciji in uporabniški izkušnji, kjer je bilo poudarjeno, da *frontend* v DeFi aplikaciji nima le vizualne, temveč tudi pomembno funkcionalno in varnostno vlogo. V praktičnem delu je bilo zato preverjano delovanje ključnih modulov, kot so povezava s kripto denarnico, zamenjava žetonov, upravljanje likvidnosti, *staking*, *yield farming* in prikaz uporabniških podatkov. Poleg avtomatiziranih testov je bilo izvedeno tudi ročno *smoke* preverjanje v lokalnem razvojnem okolju, kjer je bilo preverjeno dejansko delovanje aplikacije v povezavi z MetaMask denarnico, lokalnim Hardhat omrežjem, *backend read-only* slojem ter transakcijami nad pametnimi pogodbami.

Posebna pozornost je bila namenjena tudi elementom, ki vplivajo na varnost in preglednost uporabe, kot so opozorila pri napačnem omrežju, prikaz tolerance zdrsa cene ter ločitev med odobritvijo porabe žetonov in dejansko izvedbo funkcije. Takšno testiranje je pomembno predvsem zato, ker uporabniški vmesnik v decentraliziranih aplikacijah neposredno vpliva na pravilno razumevanje transakcij in s tem na varnost uporabnika.

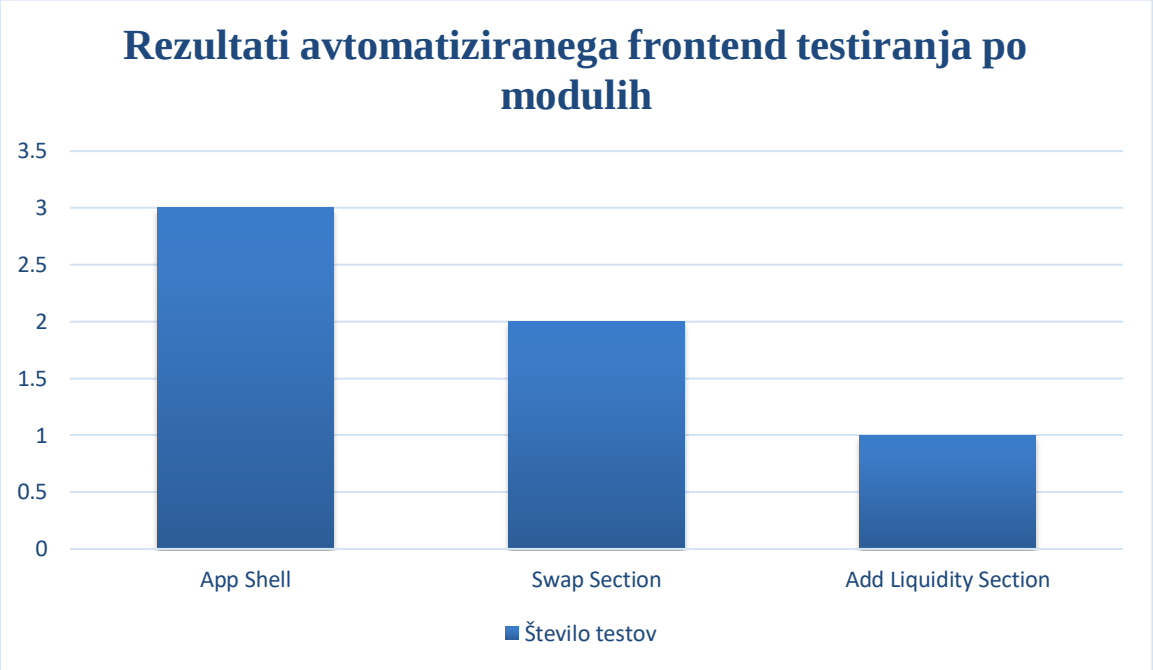
Pregled najpomembnejših preverjanj *frontend* funkcionalnosti je prikazan v tabeli 3.

Tabela 3: Pogled testiranja *frontend* funkcionalnosti

Modul	**Način preverjanja**	**Preverjene funkcionalnosti**	**Rezultat**
App shell	Avtomatizirano	*Disconnect state, wrong network state, dashboard state*	Uspešno
Swap modul	Avtomatizirano + ročno	*Approval vs action, expected output, minimum received, slippage* opozorilo	Uspešno
Add Liquidity modul	Avtomatizirano + ročno	*Approval* za oba žetona, *LP preview, projected share*	Uspešno
Staking modul	Ročno *smoke* preverjanje	*Stake, unstake, claim rewards*	Uspešno
Yield Farming modul	Ročno *smoke* preverjanje	*Approve LP, stake LP, claim rewards, unstake LP*	Uspešno
Wallet in Web3 povezava	Avtomatizirano + ročno	*Wallet connect, chainId check, switch network.*	Uspešno
Backend analytics prikaz	Ročno *smoke* preverjanje	*Makret snapshot, analytics snapshot*	Uspešno

(Vir: Lastni vir)

Rezultati avtomatiziranega *frontend* testiranja po posameznih modulih so prikazani na stolpničnem grafikonu 3.



Grafikon 3: Rezultati avtomatiziranega *frontend* testiranja po modulih

(Vir: Lastni vir)

Na sliki 29 je prikazan primer avtomatiziranega testiranja komponente uporabniškega vmesnika z uporabo ogrodja Vitest in knjižnice @testing-library/react.

```
import { fireEvent, render, screen } from '@testing-library/react';
import { describe, expect, it, vi } from 'vitest';
import { SwapSection } from './SwapSection';

describe('SwapSection', () => {
    it('shows approval action separately from swap action', () => {
        render(
            <SwapSection
                tokenInLabel="Token A"
                tokenOutLabel="Token B"
                inputBalanceFormatted="100"
                swapAllowanceFormatted="0"
                expectedAmountOutFormatted="9.09"
                minAmountOutFormatted="9.04"
                poolPriceDisplay="1 Token A ≈ 1.0000 Token B"
                estimatedGasFormatted="153421"
                poolAddress="0xpool"
                isHighSlippage={false}
                swapDirection="A_TO_B"
                onSwapDirectionChange={vi.fn()}
                swapAmountIn="10"
                onSwapAmountInChange={vi.fn()}
                isAmountInValid={true}
                swapSlippagePercent="0.5"
                onSwapSlippagePercentChange={vi.fn()}
                isSlippageValid={true}
                swapNeedsApproval={true}
                canSwap={false}
                onApproveSwapToken={vi.fn()}
                onSwap={vi.fn()}
                isSwapApprovePending={false}
                isSwapApproveConfirming={false}
                isSwapPending={false}
                isSwapConfirming={false}
            />,
        );
```

Slika 29: Primer avtomatiziranega testiranja uporabniškega vmesnika v *React* okolju

(Vir: Lastni vir)

6.3 *Uporabniško testiranje*

Uporabniško testiranje je bilo namenjeno preverjanju razumljivosti, preglednosti in praktične uporabnosti razvite aplikacije z vidika končnega uporabnika. Namen tega dela testiranja ni bil le ugotoviti, ali sistem tehnično deluje, ampak tudi preveriti, kako uporabniki razumejo posamezne funkcionalnosti in kako učinkovito jih lahko uporabljajo v praksi.

Ta del se navezuje na teoretična poglavja o uporabniški izkušnji in zasnovi uporabniškega vmesnika, kjer je bilo poudarjeno, da mora DeFi aplikacija uporabniku jasno predstaviti funkcionalnosti, parametre transakcij in morebitna tveganja. V praktičnem delu je bilo uporabniško testiranje usmerjeno v osnovne scenarije uporabe, kot so povezava s kripto denarnico, zamenjava žetonov, oddajanje likvidnosti, *staking* in uporaba *yield farming* funkcionalnosti.

Uporabniško testiranje je bilo izvedeno v omejenem obsegu kot informativno preverjanje razumljivosti, preglednosti in praktične uporabnosti razvite aplikacije z vidika končnega

uporabnika. V testiranje je bilo vključenih treh uporabnikov, ki so izvajali osnovne scenarije uporabe, kot so povezava s kripto denarnico, zamenjava žetonov, dodajanje likvidnosti, *staking* in uporaba *yield farming* funkcionalnost. Pri preverjanju so bili spremljani predvsem uspešnost izvedbe nalog, čas izvedbe nalog, število napak pri uporabi ter splošna ocena razumljivosti uporabniškega vmesnika. Takšen pristop je omogočil osnovni vpogled v to, ali aplikacija uporabnika dovolj jasno vodi skozi posamezne korake ter kateri deli vmesnika zahtevajo dodatno poenostavitev ali izrazitejša opozorila.

Pregled izvedenih uporabniških nalog in spremljanih kazalnikov je prikazan v tabeli 4.

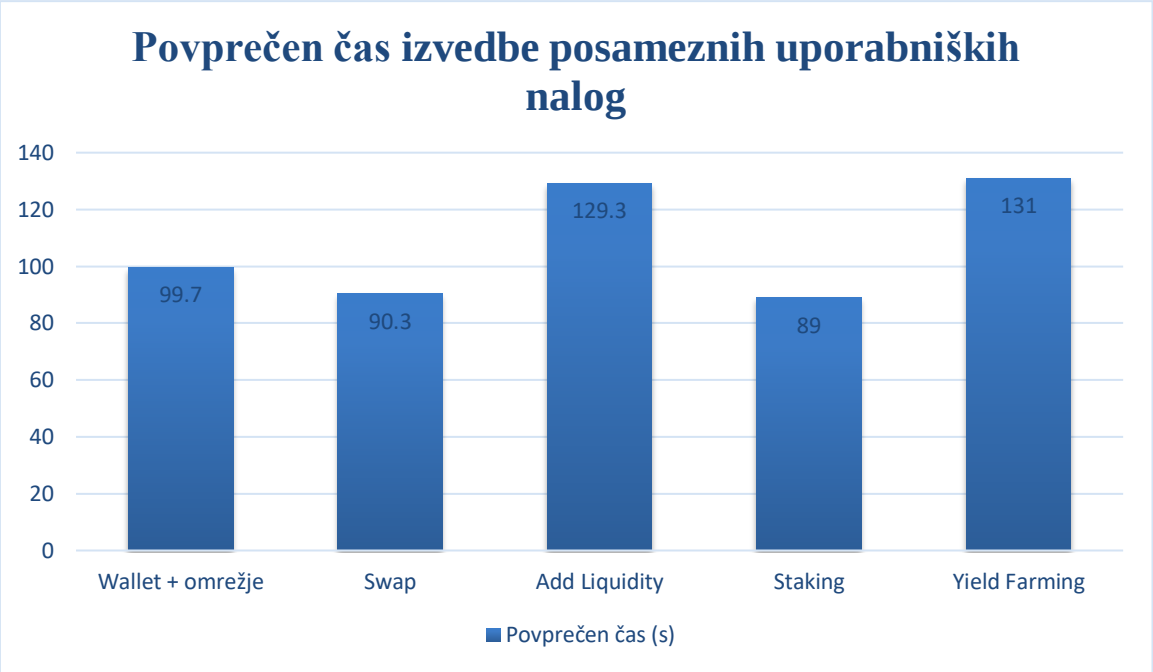
Tabela 4: Pregled scenarijev uporabniškega testiranja

Uporabnik	Naloga	Rezultat	Čas izvedbe (s)	Število napak	Ocena razumljivosti (1–5)	Opomba
U1	*Wallet* + omrežje	Uspešno	42	0	5	Takoj razumel korake, brez težav
U1	*Swap*	Uspešno	35	0	5	Vmesnik intuitiven, transakcija hitro potrjena
U1	*Add Liquidity*	Uspešno	58	0	4	Manjša nejasnost pri razmerju *tokenov*, hitro razrešena
U1	*Staking*	Uspešno	40	0	5	Jasen prikaz nagrad, postopek brez zapletov
U1	*Yield Farming*	Uspešno	52	0	5	Všeč mu je bil pregledni prikaz APY
U2	*Wallet* + omrežje	Uspešno	75	1	3	Zamenjava omrežja ni bila dovolj očitna
U2	*Swap*	Uspešno	88	1	3	Ni razumel razlike med možnostmi za *slippage*
U2	*Add Liquidity*	Uspešno	115	2	2	Zmedel ga je prikaz cenovnega razpona likvidnosti
U2	*Staking*	Uspešno	65	0	4	Postopek jasen, a gumb za

						potrditev premajhen
U2	*Yield Farming*	Uspešno	98	1	2	Ni razumel koncepta LP-*tokenov* za *farming*
U3	*Wallet* + omrežje	Neuspešno	182	4	2	Ni mu uspelo zamenjati omrežja, potreboval pomoč
U3	*Swap*	Uspešno	148	3	2	Večkrat kliknil napačen gumb, zmeden z decimalno vrednostjo
U3	*Add Liquidity*	Neuspešno	215	5	1	Ni razumel pojma »*pool share*«, nalogo opustil
U3	*Staking*	Uspešno	162	2	2	Dolgo iskal gumb za *stake*, navigacija ni intuitivna
U3	*Yield Farming*	Neuspešno	243	6	1	Prenehal po 4 minutah, preveč korakov brez pojasnil

(Vir: Lastni vir)

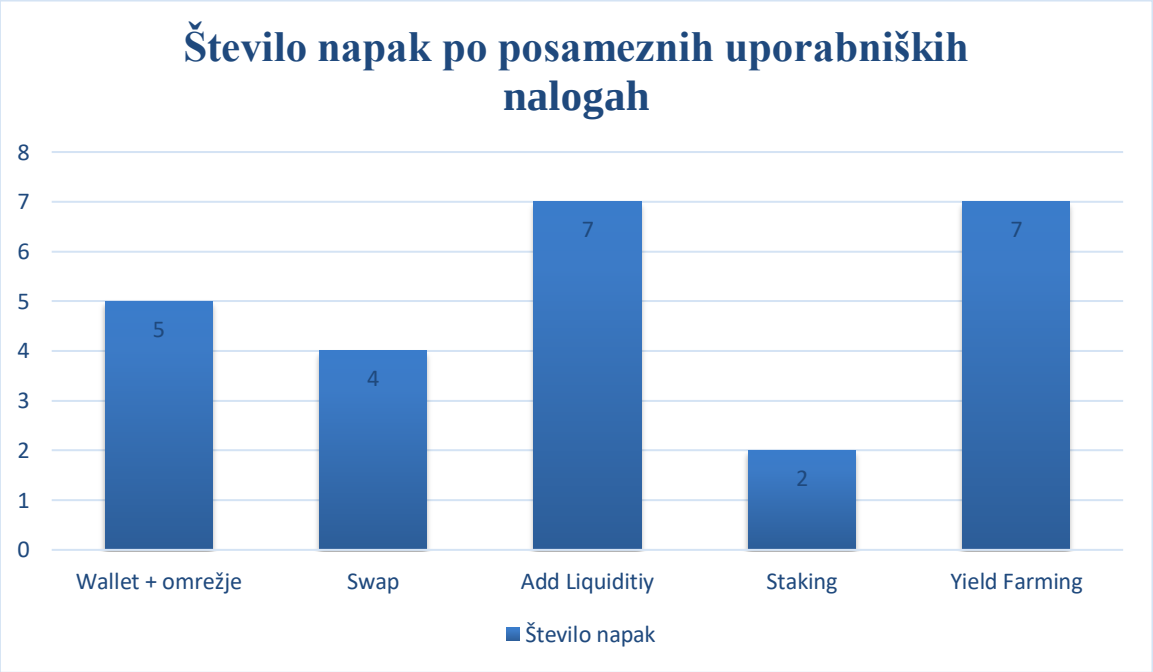
Povprečen čas izvedbe posameznih uporabniških nalog je prikazan na stolpničnem grafikonu 4.



Grafikon 4: Povprečni čas izvedbe posameznih uporabniških nalog

(Vir: Lastni vir)

Skupno število zaznanih napak pri izvedbi posameznih uporabniških nalog je prikazano na stolpničnem grafikonu 5.



Grafikon 5: Število napak po posameznih uporabniških nalogah

(Vir: Lastni vir)

Iz rezultatov, prikazanih na grafu 4, je razvidno, da so uporabniki najhitreje opravili nalogi zamenjave žetonov in *stakinga*, medtem ko sta bili časovno najzahtevnejši nalogi dodajanja likvidnosti in *yield farming*. To kaže, da sta bila modula za osnovne transakcije in *staking* uporabnikom razumljivejša, medtem ko sta bila postopka, povezana z uporabljanjem likvidnosti in uporabo LP-žetonov, najzahtevnejša za razumevanje in izvedbo.

Grafikon 5 dodatno potrjuje, da je bilo največ napak zaznanih pri nalogah dodajanja likvidnosti in *yield farminga*, najmanj pa pri *stakingu*. Takšen rezultat je pričakovan, saj omenjena modula vključujeta več korakov, dodatne koncepte, kot so LP-žetoni, *projected share* in *approval* koraki, ter zahtevata boljše razumevanje medsebojne povezanosti DeFi funkcionalnosti. Rezultati uporabniškega testiranja tako kažejo, da je aplikacija pri osnovnih funkcionalnostih dovolj razumljiva, medtem ko bi bilo pri naprednejših modulih smiselno dodatno izboljšati uporabniško vodenje, pojasnila in jasnost posameznih korakov.

6.4 Varnostno testiranje (audit, analiza napak)

Varnostno testiranje je bilo usmerjeno v preverjanje, ali razvita aplikacija upošteva ključna načela varnega razvoja DeFi sistemov. Ker aplikacija vključuje pametne pogodbe, uporabniški vmesnik in komunikacijo z blockchain omrežjem, je bilo treba preveriti ne le pravilnost delovanja, ampak tudi morebitne ranljivosti, ki bi lahko vplivale na varnost sredstev ali zanesljivost sistema.

Ta del se neposredno navezuje na teoretična poglavja o kibernetski varnosti v DeFi aplikacijah, kjer so bile obravnavane ranljivosti pametnih pogodb, *frontend* tveganja, varnost komunikacije med sloji sistema ter priporočene varnostne prakse. V praktičnem delu je bilo zato varnostno testiranje usmerjeno v pregled implementiranih zaščitnih mehanizmov, analizo morebitnih opozoril orodij za pregled kode ter preverjanje, ali aplikacija pravilno obravnava kritične scenarije.

Posebna pozornost je bila namenjena zaščiti pred ponovnim vstopom, preverjanju vhodnih podatkov, kontroli dostopa, pravilni ločitvi *approval* in *action* transakcij ter osnovni zanesljivosti uporabniškega vmesnika pri prikazu pomembnih opozoril. Takšno testiranje omogoča boljši vpogled v dejansko stopnjo varnosti aplikacije in predstavlja pomembno osnovo za končno oceno stabilnosti razvite rešitve.

Pregled ključnih varnostnih mehanizmov in ugotovitev varnostnega preverjanja je prikazan v tabeli 5.

Tabela 5: Ključni varnostni mehanizmi

Področje	Mehanizem/preverjanje	Implementacija	Ugotovitev
Pametne pogodbe	Validacija vhodnih podatkov	*Require* pogoji in *custom errors*	Prisotno
Pametne pogodbe	Zaščita pred *reentrancy*	*nonReentrant* pri ključnih funkcijah	Prisotno
Pametne pogodbe	Varno delo z ERC-20 žetoni	SafeERC20	Prisotno
Pametne pogodbe	Omejitev izplačil nagrad	*Reward pool* validacija	Prisotno
Logika DEX	*Slippage* zaščita	*minAmountOut*	Prisotno
Frontend	W*rong network* opozorilo	*chainId* preverjanje	Prisotno
Frontend	Ločitev *approval* in *action*	Ločeni gumbi in stanja	Prisotno
Frontend	Prikaz transakcijskih parametrov	Output, minimum received, slippage	Prisotno
Backend	*Read-only* sloj brez podpisovanja	Brez upravljanja privatnih ključev	Prisotno
Varnostna analiza	Statični pregled kode	*Slither*	Brez kritičnih ranljivosti

(Vir: Lastni vir)

6.5 *Analiza rezultatov in primerjava z obstoječimi rešitvami*

Na podlagi rezultatov testiranja je mogoče oceniti, da razvita aplikacija dosega glavne funkcionalne cilje, zastavljene v praktičnem delu diplomskega dela. *Unit* testi pametnih pogodb so potrdili pravilno delovanje ključne logike za upravljanje žetonov, zamenjavo sredstev, dodajanje in odstranjevanje likvidnosti, *staking* ter *yield farming*. *Frontend* testiranje je pokazalo, da uporabniški vmesnik pravilno prikazuje podatke, vodi uporabnika skozi *approval* in *action* korake ter vključuje osnovne varnostne elemente, kot so preverjanje pravilnega omrežja, prikaz tolerance zdrsa cene in ločitev med odobritvijo ter dejansko transakcijo. Tudi uporabniško testiranje je pokazalo, da so osnovne funkcionalnosti aplikacije razumljive, medtem ko so kompleksnejši postopki, povezani z likvidnostjo in LP-žetoni, za uporabnike zahtevnejši.

Razvita rešitev po funkcionalnem obsegu sledi osnovnim vzorcem sodobnih DeFi aplikacij, vendar ostaja prototipna implementacija. V primerjavi z uveljavljenimi rešitvami, kot so Uniswap, aplikacija vključuje glavne gradnike decentraliziranih financ, vendar v poenostavljeni obliki. Referenčne rešitve vključujejo širši nabor funkcionalnosti, naprednejšo optimizacijo,

večjo količino analitičnih podatkov, integracijo z javnimi omrežji ter višjo stopnjo produkcijske pripravljenosti. Kljub temu razvita aplikacija uspešno izpolnjuje svoj namen, saj na preglednem in tehnično delujočem primeru pokaže povezovanje pametnih pogodb, *frontend* aplikacije, povezovanje s kripto denarnico in podporni *backend* sloj.

Primerjava razvite aplikacije z izbranimi referenčnimi DeFi rešitvami je prikazana v tabeli 6.

Tabela 6: Primerjava razvite aplikacije z izbranimi referenčnimi DeFi rešitvami

Vidik	Razvita aplikacija	Referenčne DeFi rešitve	Opomba
Zamenjava žetonov	Da	Da	Referenčne rešitve običajno vključujejo naprednejši *routing* in globljo likvidnost
Dodajanje in odstranjevanje likvidnosti	Da	Da	V prototipu je funkcionalnost poenostavljena in lokalno testirana
Staking	Da	Delno	V prototipu je *staking* ločen modul
Yield Farming	Da	Delno	Implementirano z LP žetoni in dodatnimi nagradami
Povezava z denarnico	Da	Da	Podprta je Web3 integracija z MetaMask
Backend analytics sloj	Osnovni	Napreden	Referenčne rešitve praviloma vključujejo obsežnejše podatke in zgodovinske metrike
Varnostni mehanizmi	Osnovni zaščitni mehanizmi	Napredni in večkrat *auditirani*	Prototip vključuje ključne zaščite, ni pa produkcijska aplikacija
Produkcijska pripravljenost	Ne	Da	Razvita aplikacija je prototipna rešitev za demonstracijo koncepta

(Vir: Lastni vir)

Na podlagi izvedene analize je mogoče ugotoviti, da razvita aplikacija uspešno demonstrira ključne principe delovanja DeFi sistema in potrjuje tehnično izvedljivost zasnovane arhitekture. Kljub omejitvam, ki so značilne za prototipne rešitve, aplikacija vključuje bistvene funkcionalnosti, osnovne varnostne mehanizme in delujočo povezavo med vsemi glavnimi sloji sistema. Takšni rezultati predstavljajo ustrezno osnovo za preverjanje hipotez in oblikovanje zaključnih ugotovitev diplomskega dela.

6.6 Odgovori na raziskovalna vprašanja in preverjanje hipotez

V diplomskem delu sem si zastavil več raziskovalnih vprašanj, ki so bila povezana z uporabno vrednostjo DeFi funkcionalnosti, integracijo med posameznimi sloji sistema, vplivom uporabniške izkušnje na zaupanje uporabnikov ter vlogo varnostnih praks pri razvoju decentralizirane aplikacije. Na podlagi izvedenega razvoja in testiranja lahko na ta vprašanja podam naslednje odgovore.

Na prvo raziskovalno vprašanje, kako vključitev ključnih funkcionalnosti decentraliziranih financ vpliva na uporabno vrednost in celovitost aplikacije, lahko odgovorim, da vključitev funkcionalnosti, kot so decentralizirane izmenjava, *staking* in *yield farming*, bistveno poveča uporabno vrednost sistema. Namesto ene same ozko usmerjene funkcije aplikacija uporabniku omogoča več različnih načinov interakcije s sredstvi, kar povečuje funkcionalno celovitost rešitve. Prav ta večplastnost predstavlja eno izmed glavnih značilnosti sodobnih DeFi aplikacij.

Na drugo raziskovalno vprašanje, kako pravilna arhitekturna integracija pametnih pogodb z uporabniškim vmesnikom in zalednim sistemom prispeva k zmanjšanju napak in omogoča skalabilno delovanje, lahko odgovorim, da je ravno jasna porazdelitev sistema na blockchain sloj, *frontend*, Web3 integracijo in podporni *backend* omogočila bolj pregledno implementacijo ter lažje testiranje. Modularna zasnova je zmanjšala kompleksnost posameznih komponent, poenostavila povezovanje celotnega sistema in omogočila učinkovitejše odpravljanje napak v razvojni fazi.

Na tretje raziskovalno vprašanje, kako uporabniška izkušnja in transparentnost transakcij vplivata na zaupanje ter angažiranost uporabnikov, ugotavljam, da ima uporabniški vmesnik v DeFi okolju pomembno vlogo ne le z vidika uporabnosti, temveč tudi z vidika varnosti. Jasna ločitev med *approval* in *action* koraki, prikaz tolerance zdrsa cene, minimalnega prejetega zneska ter opozorila ob napačnem omrežju prispevajo k boljši preglednosti sistema in zmanjšujejo verjetnost napačnih uporabniških odločitev. Uporabniško testiranje je pokazalo, da uporabniki bolje razumejo osnovne funkcionalnosti, medtem ko je pri zahtevnejših modulih potrebna dodatna poenostavitev.

Na četrto raziskovalno vprašanje, v kolikšni meri implementacija kibernetskih varnostnih praks zmanjšuje tveganje za uspešne napade na DeFi aplikacijo, lahko odgovorim, da uporaba osnovnih varnostnih mehanizmov bistveno izboljša zanesljivost sistema. V praktičnem delu so bili implementirani pomembni zaščitni ukrepi, kot so *reentrancy* zaščita, validacija vhodnih

podatkov, uporaba preverjenih knjižnic, *slippage* zaščita, *wrong network* opozorila ter *read-only backend* pristop. Varnostna analiza s statičnim pregledom kode ni pokazala kritičnih ranljivosti, kar potrjuje, da dosledno upoštevanje varnostnih načel pomembno zmanjša tveganje napadov, čeprav ga v produkcijskem okolju nikoli ne odpravi popolnoma.

Na peto raziskovalno vprašanje, kateri so ključni arhitekturni in tehnološki izzivi pri razvoju prototipne DeFi spletne aplikacije na omrežju Ethereum, lahko odgovorim, da so največji izzivi povezani z usklajevanjem več medsebojno odvisnih slojev sistema, zagotavljanjem varne komunikacije med njimi ter predstavitvijo kompleksnih DeFi konceptov na uporabniku razumljiv način. Posebej zahtevni so bili povezovanje pametnih pogodb s *frontendom*, pravilna implementacija *approval* logike, varnostno preverjanje pogodb in uporabniška razumljivost modulov, kot sta likvidnostni bazen in *yield farming*.

Na podlagi odgovorov na raziskovalna vprašanja lahko ovrednotim tudi zastavljene hipoteze.

Hipoteza H1 je bila delno potrjena. Diplomsko delo sicer izhaja iz vključitve ključnih DeFi funkcionalnosti, kot so DEX, *staking* in *yield farming*, pri čemer aplikacija s tem nedvomno pridobi večjo uporabno vrednost. Ker pa popolna implementacija posojanja ni bila vključena v praktični del, hipoteze ni mogoče v celoti potrditi.

Hipoteza H2 je bila potrjena. Pravilna integracija pametnih pogodb z uporabniškim vmesnikom in podpornim *backend* slojem je omogočila preglednejšo strukturo sistema, lažje testiranje in zmanjšanje napak v razvoju, kar potrjuje pomen ustrezne arhitekturne zasnove.

Hipoteza H3 je bila potrjena. Jasna predstavitev transakcijskih parametrov, opozorila pri napačnem omrežju in pregledna ločitev posameznih korakov potrjujejo, da uporabniška izkušnja in transparentnost pomembno vplivata na zaupanje uporabnika v sistem, čeprav so naprednejše funkcionalnosti za manj izkušene uporabnike še vedno zahtevnejše.

Hipoteza H4 je bila potrjena. Implementacija osnovnih varnostnih praks, tako na ravni pametnih pogodb kot na ravni uporabniškega vmesnika in podpornega *backend* sloja, je pomembno prispevala k zmanjšanju tveganja napak in potencialnih zlorab v razvitem prototipu.

6.7 Prednosti, slabosti in možnosti izboljšav spletne aplikacije

Med glavne prednosti razvite spletne aplikacije sodi njena modularna arhitektura, saj so pametne pogodbe, *frontend*, Web3 integracija in *backend* jasno funkcionalno razmejeni. Takšna

zasnova izboljšuje preglednost implementacije, omogoča lažje testiranje posameznih delov sistema in olajša nadaljnje razširjanje aplikacije. Druga pomembna prednost je vključitev več ključnih DeFi funkcionalnosti v enoten prototip, saj aplikacija omogoča menjavo žetonov, upravljanje likvidnosti, *staking* in *yield farming*. Pomembna prednost je tudi upoštevanje osnovnih varnostnih mehanizmov, kot so zaščita pred *reentrancy*, validacija vhodnih podatkov, uporaba preverjenih knjižnic in varnostni elementi v uporabniškem vmesniku.

Kljub temu ima razvita aplikacija tudi več omejitev. Prva pomembna slabost je, da gre za prototipno rešitev, ki ni namenjena produkcijski uporabi. Aplikacija je bila razvita in preverjena v testnem okolju, brez zunanje varnostne revizije in brez dejanske uporabe na produkcijskem omrežju. Druga omejitev je povezana z uporabniško izkušnjo pri naprednejših funkcionalnostih. Rezultati uporabniškega testiranja so pokazali, da so moduli, kot sta dodajanje likvidnosti in *yield farming*, za manj izkušene uporabnike težje razumljivi, saj vključujejo dodatne koncepte, kot so LP-žetoni, *projected share* in večstopenjska *approval* logika. Tretja slabost je omejen obseg *backend* analitike, ki trenutno služi predvsem osnovnim pregledom podatkov in ne vključuje naprednejših metrik, zgodovinskih analiz ali širše primerjalne obdelave podatkov.

Možnosti izboljšav so zato predvsem na treh področjih. Prvo področje je izboljšanje uporabniške izkušnje, predvsem z dodatnimi pojasnili, izrazitejšimi opozorili in poenostavljenim vodenjem uporabnika skozi kompleksnejše korake. Drugo področje je razširitev analitičnega dela *backend* sloja, kjer je bilo mogoče vključiti naprednejše metrike, zgodovino transakcij, podrobnejši portfeljski pregled in natančnejše izračune. Tretje področje pa predstavlja dodatna varnostna utrditev, zlasti v smeri zunanjega *audita*, širše statične in dinamične analize ter utrditev *deployment* in CI/CD procesa.

6.8 Predlogi za nadaljnji razvoj

Razvita aplikacija predstavlja dobro osnovo za nadaljnji razvoj v smeri celovitejše in produkcijsko zrele DeFi rešitve. Eden od prvih logičnih korakov nadaljnjega razvoja bi bil prehod iz lokalnega testnega okolja na javno testno omrežje in pozneje, ob ustreznih dodatnih varnostnih preverjanjih, tudi na produkcijsko omrežje. To bi omogočilo bolj realistično preverjanje delovanja v dejanskem blockchain okolju ter dodatno testiranje uporabniške izkušnje v realnejših pogojih.

Naslednji pomemben korak bi bila razširitev funkcionalnega obsega aplikacije. Ker je v teoretičnem delu obravnavano tudi področje posojanja in izposoje sredstev, bi bila smiselna nadgradnja sistema z modulom za *lending* in *borrowing*, kar bi dodatno povečalo funkcionalno celovitost aplikacije. Poleg tega bi bilo mogoče razširiti tudi DEX logiko, na primer z naprednejšim *routanjem*, natančnejšo podporo več likvidnostnim parom ali dodatnimi mehanizmi za izračun provizij in nagrad.

Velik potencial za nadaljnji razvoj predstavlja tudi izboljšanje uporabniškega vmesnika. Pri prihodnjih nadgradnjah bi bilo smiselno vključiti dodatne razlage posameznih pojmov, vizualno vodenje uporabnika po korakih, bolj intuitivno predstavitev LP-žetonov, izrazitejša varnostna opozorila in naprednejše elemente za prikaz analitike. Takšne izboljšave bi pripomogle k temu, da bi aplikacija postala dostopnejša tudi manj izkušenim uporabnikom.

Na področju varnosti bi bil najpomembnejši korak izvedba varnostnega pregleda pametnih pogodb in širša uporaba avtomatiziranih orodij za analizo odvisnosti, infrastrukture in *deployment* procesa. Smiselno bi bilo tudi utrditi upravljanje konfiguracij, dodati več nivojev zaščite pri *deploymentu* ter nadgraditi spremljanje napak in dogodkov v *backend* sloju.

Na podlagi izvedenega dela lahko zaključim, da razvita spletna aplikacija predstavlja uspešno prototipno osnovo za nadaljnje raziskovanje, izboljševanje in razširjanje decentraliziranih finančnih rešitev. Hkrati pa potrjuje, da je pri razvoju DeFi sistemov poleg tehnične funkcionalnosti enako pomembno tudi razumevanje varnosti, arhitekture in uporabniške izkušnje.

7 SKLEP

V diplomskem delu sem obravnaval razvoj decentralizirane finance (DeFi) spletne aplikacije, pri čemer je bil glavni namen povezati teoretična izhodišča decentraliziranih financ s praktično implementacijo delujoče prototipne rešitve. Delo je pokazalo, da področje DeFi temelji na več medsebojno povezanih tehnoloških in varnostnih elementih, med katerimi imajo osrednjo vlogo blockchain tehnologija, pametne pogodbe, standardi žetonov, Web3 integracija, uporabniški vmesnik in varnostni mehanizmi. Pri razvoju takšne aplikacije zato ni dovolj zgolj implementirati posamezne funkcionalnosti, temveč je treba celoten sistem načrtovati kot povezano, pregledno in varnostno premišljeno arhitekturo.

V teoretičnem delu so bili predstavljeni ključni koncepti, ki omogočajo razumevanje delovanja decentraliziranih finančnih aplikacij. Poseben poudarek je bil namenjen omrežju Ethereum, pametnim pogodbam, standardu ERC-20, decentraliziranim borzam, *stakingu*, *yield farmingu* ter varnostnim tveganjem, ki so značilna za DeFi okolje. Ugotovil sem, da DeFi aplikacije uporabnikom omogočajo večjo samostojnost in neposreden nadzor nad sredstvi, hkrati pa zaradi odsotnosti centralnega posrednika prinašajo večjo odgovornost za pravilnost kode, varnost uporabniške interakcije in zanesljivost celotnega sistema.

V praktičnem delu je bila razvita prototipna spletna aplikacija, ki združuje več ključnih funkcionalnosti decentraliziranih financ. Aplikacija vključuje žetonsko pogodbo ERC-20, pogodbo za likvidnostni bazen z logiko DEX, *staking* pogodbo, *yield farming* pogodbo, *frontend* v okolju React in TypeScript, Web3 povezavo s kripto denarnico ter podporni *backend* sloj za pripravo preglednih podatkov. S tem je bil dosežen glavni cilj diplomskega dela, saj je bila razvita rešitev, ki ne ostane zgolj na ravni teoretičnega opisa, temveč praktično prikazuje delovanje več DeFi mehanizmov v enotnem sistemu.

Rezultati testiranja so pokazali, da razviti prototip pravilno izvaja glavne funkcionalnosti aplikacije. *Unit* testi pametnih pogodb so potrdili pravilno delovanje logike za upravljanje žetonov, dodajanje in odstranjevanje likvidnosti, zamenjavo sredstev, *staking* ter *yield farming*. *Frontend* testiranje je pokazalo, da uporabniški vmesnik pravilno vodi uporabnika skozi ključne korake, kot so povezava denarnice, odobritev porabe žetonov, izvedba transakcije in prikaz rezultatov. Uporabniško testiranje pa je pokazalo, da so osnovne funkcionalnosti uporabnikom razumljivejše, medtem ko zahtevnejši moduli, povezani z likvidnostjo in LP-žetoni, zahtevajo dodatna pojasnila in boljše uporabniško vodenje.

Pomembna ugotovitev diplomskega dela je, da varnost v DeFi aplikacijah ne sme biti obravnavana kot dodatek po končani implementaciji, temveč kot sestavni del celotnega razvojnega procesa. Pri implementaciji so bili zato uporabljeni varnostni mehanizmi, kot so validacija vhodnih podatkov, zaščita pred *reentrancy* napadi, ločevanje odobritvenih in izvedbenih transakcij, preverjanje pravilnega omrežja, uporaba preverjenih knjižnic ter omejevanje vloge *backend* sloja na podporne funkcije. Ti ukrepi ne morejo popolnoma odpraviti vseh tveganj, vendar pomembno zmanjšujejo možnost napak in zlorab v prototipnem okolju.

Na podlagi izvedenega dela lahko sklenem, da je mogoče tudi v okviru prototipne rešitve uspešno povezati več temeljnih DeFi funkcionalnosti v enoten in delujoč sistem. Razvita aplikacija potrjuje, da pravilna modularna arhitektura, smiselna uporaba pametnih pogodb, pregledna Web3 integracija in jasen uporabniški vmesnik pomembno prispevajo k zanesljivosti in uporabnosti decentralizirane finančne aplikacije. Hkrati rezultati kažejo, da je pri razvoju takšnih rešitev treba posebno pozornost nameniti razumevanju uporabnika, saj so nekateri DeFi koncepti za manj izkušene uporabnike zahtevni in lahko vodijo do napačnih odločitev.

Razvita aplikacija ne predstavlja produkcijsko pripravljene finančne platforme, temveč prototipno rešitev, namenjeno prikazu koncepta, preverjanju arhitekture in razumevanju ključnih izzivov razvoja DeFi sistemov. Za nadaljnji razvoj bi bilo smiselno aplikacijo najprej dodatno preveriti na javnem testnem omrežju, razširiti analitični del, izboljšati uporabniško izkušnjo pri kompleksnejših modulih, vključiti naprednejše varnostne preglede ter v prihodnosti dodati tudi funkcionalnosti posojanja in izposoje sredstev. S tem bi se aplikacija lahko razvila v celovitejšo rešitev, ki bi še bolje prikazovala širino in uporabno vrednost decentraliziranih financ.

Diplomsko delo tako potrjuje, da razvoj DeFi spletne aplikacije zahteva povezovanje znanja s področja programiranja, blockchain tehnologije, kibernetske varnosti, uporabniške izkušnje in sistemske arhitekture. Prav ta povezanost različnih področij predstavlja največji izziv, hkrati pa tudi največjo vrednost razvite rešitve.

8 VIRI IN LITERATURA

Anglen, J. (n. d.). *Smart Contracts and DeFi: Transforming Decentralized Finance*. Pridobljeno iz Rapid Innovation Post: https://www.rapidinnovation.io/post/smart-contracts-and-defi-transforming-decentralized-finance

Beyondi. (13. avgust 2024). *Top 7 Backend Strategies for Enhancing Security in DeFi Applications*. Pridobljeno iz Beyondi | The Web3 Company: https://blog.beyondi.co/top-7-backend-strategies-for-enhancing-security-in-defi-applications-2c583c935aea

Bomberbot. (20. april 2024). *Decentralized Applications Architecture: Back End, Security and Design Patterns*. Pridobljeno iz Bomberbot blockchain: https://www.bomberbot.com/blockchain/decentralized-applications-architecture-back-end-security-and-design-patterns/

Cholakov, S. (16. januar 2025). *DeFi Front-End Exploits: Security Attacks & Threat Detection*. Pridobljeno iz Three Sigma Blog: https://threesigma.xyz/blog/exploit/defi-front-end-exploits

CointTracker. (n. d.). *What is an RPC node? The gateway to blockchain networks*. Pridobljeno iz CoinTracker Learn: https://www.cointracker.io/learn/remote-procedure-call-rpc

CryptoCloud. (25. april 2025). *What is dex crypto - guide*. Pridobljeno iz CryptoCloud Blog: https://cryptocloud.plus/en/blog/what-is-dex-crypto-guide

DWELLIR. (10. november 2025). *Top 10 Polygon RPC providers powering high-performance Web3 apps in 2025*. Pridobljeno iz Dwellir Blog: https://www.dwellir.com/blog/10-best-polygon-rpc-providers-2025

DWELLIR. (17. november 2025). *Top 6 Base RPC Providers 2025: Transparent Pricing & Performance Comparison*. Pridobljeno iz Dwellir Blog: https://www.dwellir.com/blog/top-6-base-providers-2025

Ethereum. (25. februar 2026). *Ethereum Virtual Machine (EVM)*. Pridobljeno iz Ethereum developer: https://ethereum.org/developers/docs/evm/

Ethereum Whitepaper. (22. april 2026). Pridobljeno iz Ethereum: https://ethereum.org/whitepaper/

Gate. (n. d.). *What Is Ethers.js? A Deep Dive Into Ethereum's JavaScript Library*. Pridobljeno iz Gate Crypto Wiki: https://www.gate.com/crypto-wiki/article/what-is-ethers-js-a-deep-dive-into-ethereum-s-javascript-library-20260208

Hacken. (26. junij 2025). *Defi Staking*. Pridobljeno iz Hacken Discover: https://hacken.io/discover/defi-staking/

Hayes, A. (8. oktober 2025). *Understanding Slippage in Finance: Key Insights and Examples*. Pridobljeno iz Investopedia Terms: https://www.investopedia.com/terms/s/slippage.asp

Kraken Learn Team. (22. april 2025). *What is yield farming*. Pridobljeno iz Kraken Learn: https://www.kraken.com/learn/what-is-yield-farming

Kramer, N. (25. april 2025). *Daily.dev*. Pridobljeno iz Daily.dev Blog: https://daily.dev/blog/5-steps-to-build-defi-apps

Marchesi, L., Michele, M., Pompianu, L. in Tonelli, R. (9. avgust 2020). *Decentralized Finance: On Blockchain- and Smart Contract-based Financial Markets.* doi:10.48550/arXiv.2008.04761

Meegle. (12. marec 2025). *DeFi Security Best Practices*. Pridobljeno iz Meegle topics: https://www.meegle.com/en_us/topics/web3/defi-security-best-practices

MetaMask. (n. d.). *Detect a user's network*. Pridobljeno iz MetaMask Wallet: https://docs.metamask.io/wallet/how-to/manage-networks/detect-network

NadcabAdmin. (4. junij 2025). *Dao in defi space*. Pridobljeno iz Nadcab Blog: https://www.nadcab.com/blog/dao-in-defi-space

Nakamoto, S. (n. d.). *Bitcoin: A Peer-to-Peer Electronic Cash System.* Pridobljeno iz bitcoin.org: https://bitcoin.org/bitcoin.pdf

Nundini, K. (3. julij 2025). *How To Create A DeFi App? A Guide For DeFi App Development*. Pridobljeno iz Raftlabs Blog: https://www.raftlabs.com/blog/scalable-defi-app-guide/

Orb. (9. november 2022). *General Smart Contract Security: A Guide for Web3 Development*. Pridobljeno iz Medium – Balancer protocol: https://medium.com/balancer-protocol/general-smart-contract-security-a-guide-for-web3-development-80448eca93e

OSL. (21. marec 2025). *The evolution of decentralized finance defi from lending to yield farming*. Pridobljeno iz OSL Academy – Article: https://www.osl.com/hk-

en/academy/article/the-evolution-of-decentralized-finance-defi-from-lending-to-yield-farming

OSL. (6. februar 2025). *What is solidity in crypto*. Pridobljeno iz OSL Article: https://www.osl.com/hk-en/academy/article/what-is-solidity-in-crypto

Petruk, M. (4. december 2023). *Best Practices for Building Secure Decentralized Apps*. Pridobljeno iz WeSoftYou: https://wesoftyou.com/ai/best-practices-for-building-secure-decentralized-apps/

Petruk, M. (10. julij 2023). *How Much Does It Cost to Develop a Cryptocurrency Exchange App?* Pridobljeno iz WeSoftYou – OUTSOURCING: https://wesoftyou.com/outsourcing/how-much-does-it-cost-to-build-crypto-exchange-app/

Reiff, N. (14. april 2025). *Decentralized Autonomous Organization (DAO): Definition, Purpose, and Example*. Pridobljeno iz Investopedia Tech: https://www.investopedia.com/tech/what-dao/

Reiff, N. (6. april 2025). *What Are ERC-20 Tokens on the Ethereum Network?* Pridobljeno iz Investopedia News: https://www.investopedia.com/news/what-erc20-and-what-does-it-mean-ethereum/

Ribhavmodi. (29. december 2025). *Web3 Infrastructure: RPCs, Nodes, Infura/Alchemy (The Invisible Plumbing)*. Pridobljeno iz Medium coinmonks: https://medium.com/coinmonks/web3-infrastructure-rpcs-nodes-infura-alchemy-the-invisible-plumbing-3225bfdb0cfe

Saykhov, R., Goodell, G., De Montigny, D., Schoering, M. in Treleaven, P. (24. november 2023). *Decentralized token economy theory (DeTEcT): token pricing, stability and governance for token economies*. Pridobljeno iz Frontiersin: https://www.frontiersin.org/journals/blockchain/articles/10.3389/fbloc.2023.1298330/full

Schär, F. (2021). *Decentralized Finance on Blockchain and Smart Contract-based Financial Markets.* doi:10.20955/r.103.153-74

Stellar Development Foundation. (2025). *Lending and borrowing markets*. Pridobljeno iz Stellar Learn: https://stellar.org/learn/lending-and-borrowing-markets

Tatarchenko, M. (28. april 2025). *Integrating Decentralized Finance (DeFi) Protocols in Web3*. Pridobljeno iz Sapient.pro Blog: https://sapient.pro/blog/integrating-defi-protocols-in-web3

TRM Labs. (2025). *Decentralized finance, adaptive regulatory sandboxes 1*. Pridobljeno iz TRMlabs Glossary: https://www.trmlabs.com/glossary/decentralized-finance#smart-contract-risks-1

9 PRILOGE

Priloga 1: Spletna DeFi aplikacija