VIŠJA STROKOVNA ŠOLA ACADEMIA

MARIBOR

PRIMERJALNA ANALIZA INTEGRACIJE PLAČILNEGA SISTEMA STRIPE V REACT IN NEXT.JS SPLETNI REZERVACIJSKI APLIKACIJI

Kandidat: Andrej Karničar

Vrsta študija: izredni študij

Študijski program: Računalništvo

Mentor predavatelj: mag. Ervin Schaff

Mentor v podjetju: Gregor Zebec, inž. rač. in inf.

Lektor: Ana Kovač, prof slov. in angl.

Maribor, 2026

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Podpisani Andrej Karničar, sem avtor diplomskega dela z naslovom »Primerjalna analiza integracije plačilnega sistema Stripe v React in Next.js spletni rezervacijski aplikaciji«, ki sem ga napisal pod mentorstvom mag. Ervina Schaffa.

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, 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, september 2026 Podpis študenta:

ZAHVALA

Zahvaljujem se mentorju mag. Ervinu Schaffu za strokovne usmeritve pri pripravi diplomskega dela in mentorju v podjetju Gregorju Zebcu za podporo pri praktičnem delu. Zahvaljujem se tudi vsem, ki so me med študijem spodbujali in mi pomagali pri pridobivanju znanja s področja računalništva.

POVZETEK

V diplomskem delu primerjam integracijo plačilnega sistema Stripe v aplikacijah React s Supabase in Next.js. Za študijo primera uporabim projekt Zeen Connect, v katerem obe uporabniški izvedbi povežem z istim testnim zaledjem. Raziskavo oprem na pregled kode, izvedene avtomatizirane teste, dejanske transakcije v testnem načinu Stripe in dokumentiran pregled uporabniškega vmesnika.

Pri primerjavi ločim skupno zaledno logiko od razlik v uporabniških aplikacijah. V Reactu spremljam neposredne klice iz brskalnika v Supabase Edge Functions, v Next.js pa vključitev istega zaledja prek strežniških akcij. Uspešnost presojam glede na odziv sistema ter povezane zapise plačila in rezervacije v podatkovni bazi.

Izvedem 948 lokalnih testov, osnovni plačilni preizkus, sedem razširjenih integracijskih scenarijev ter dodatne preizkuse cene, preklica in zahteve brez prijave. Na dveh končnih točkah webhook izvedem šestnajst zahtevkov za preverjanje podpisa in ločitve načinov. Pri trinajstih dodatnih dostavah istega plačilnega dogodka preverim, da se ne ustvarijo dodatni poslovni učinki. Preverim tudi dejansko testno vračilo, spremembo termina, dogodek spora ter izplačilo Connect.

Pri pregledu uporabniškega vmesnika ugotovim, da Next.js isti rezervacijski primer prikaže v dveh fazah, React pa v treh. V cenovnem povzetku najdem razliko: React prikaže 122 evrov, Next.js in zaledje pa 100 evrov. V uporabljenem brskalniškem okolju zabeležim tudi težavo pri nalaganju kartičnega okvirja. Ugotovitve podprem z dejanskimi posnetki zaslona in hevrističnim pregledom. Kot glavni rezultat izpostavim preverjeno delovanje izbranih zalednih scenarijev ter potrebo po enotnem cenovnem prikazu in jasnem ravnanju ob napakah v uporabniškem toku.

Raziskavo omejim na pregledani različici aplikacij, izbrane naloge in testno okolje. Preizkusi ne vključujejo prenosa resničnega denarja ali raziskave z uporabniki, zato ugotovitev ne posplošim na vsa okolja in uporabnike. Za sledljivost rezultatov ohranim izpise testov, strukturirane zapisnike ter zaslonske slike, ki omogočajo preverjanje ugotovitev in ponovitev preizkusov. Pri razlagi rezultatov ločim preverjena poslovna stanja od opažanj vmesnika, ki jih pridobim z lastnim hevrističnim pregledom.

Ključne besede: *Stripe, React, Next.js, Supabase, rezervacija, integracijsko testiranje, uporabnost.*

ABSTRACT

COMPARATIVE ANALYSIS OF STRIPE PAYMENT SYSTEM INTEGRATION IN REACT AND NEXT.JS WEB BOOKING APPLICATIONS

In this diploma thesis, I compare the integration of the Stripe payment system in a React application with Supabase and a Next.js application. I use the Zeen Connect project as a case study, connecting both user interfaces to the same test backend. The research is based on source code inspection, automated tests, actual Stripe test-mode transactions, and a documented review of the user interface.

In the comparison, I distinguish the shared backend logic from the differences between the user applications. In React, I examine direct browser calls to Supabase Edge Functions, while in Next.js, I examine the integration of the same backend through server actions. I assess success based on system responses and the associated payment and booking records in the database.

I conduct 948 local tests, a baseline payment test, seven extended integration scenarios, and additional tests of pricing, cancellation, and unauthenticated requests. I send sixteen requests to two webhook endpoints to verify signature validation and mode separation. For thirteen additional deliveries of the same payment event, I verify that no additional business effects are created. I also verify an actual test refund, a rescheduling proposal, a dispute event, and a Connect payout event. In the user interface review, I find that Next.js presents the same booking scenario in two stages, while React presents it in three. I identify a difference in the price summary: React displays EUR 122, while Next.js and the backend display EUR 100. I also document an issue with loading the payment card frame in the browser environment used for the review. I support the findings with actual screenshots and a heuristic review. As the main result, I highlight the verified functioning of the selected backend scenarios and the need for consistent price presentation and clear handling of errors in the user flow.

The research is limited to the reviewed application versions, selected tasks, and test environment. The tests involve neither real money transfers nor a user study, so I do not generalise the findings to all environments and users. To ensure the traceability of the results, I retain test outputs, structured records, and screenshots that enable the findings to be verified and the tests to be repeated. In interpreting the results, I distinguish verified business states from interface observations obtained through my own heuristic review.

Keywords: *Stripe, React, Next.js, Supabase, booking, integration testing, usability.*

KAZALO VSEBINE

1 UVOD 10

1.1 Opredelitev problema 10

1.2 Namen in cilji 11

1.3 Raziskovalna vprašanja in hipoteze 11

1.4 Metode in razmejitev raziskave 12

1.5 Zgradba dela 13

2 TEORETIČNA IZHODIŠČA 14

2.1 Rezervacija in plačilo 14

2.2 Checkout, PaymentIntent in vdelani obrazec 14

2.3 Cena kot strežniška odločitev 15

2.4 Webhook in pristnost dogodka 16

2.5 Ponovitve, sočasnost in idempotentnost 16

2.6 React in neposredna povezava s Supabase 17

2.7 Next.js in strežniške akcije 17

2.8 Preverjanje na različnih ravneh 18

2.9 Uporabnost in dostopnost 18

2.10 Lastništvo stanja in obnova postopka 19

3 PRIMERJALNA ZASNOVA RAZISKAVE 21

3.1 Gradivo in sledljivost različic 21

3.2 Testno okolje 22

3.3 Enak poslovni primer 22

3.4 Merila uspeha 23

3.5 Zaporedje izvedbe 24

3.6 Zajem in razvrščanje dokazov 24

3.7 Omejitve sklepanja 25

3.8 Natančnost zapisnika in preverjanje nasprotujočih si podatkov 25

4 PRAKTIČNI PREGLED IN PREIZKUS 27

4.1 Priprava in prvi zagon 27

4.2 Lokalni testi obeh projektov 27

4.3 Uspešno testno plačilo 28

4.4 Ponovna dostava dogodka 29

4.5 Zavrnitev kartice 30
4.6 Preklic in zahteva brez prijave 30
4.7 Podpisi in ločitev testnega načina 31
4.8 Vračilo, sprememba termina in Connect 31
4.9 Dejanski uporabniški tok React 32
4.10 Dejanski uporabniški tok Next.js 34
4.11 Mobilni prikaz in semantični pregled 36
4.12 Zaključek izvedbe in ohranitev rezultatov 37
5 PRIMERJALNA ANALIZA IN VREDNOTENJE 38
5.1 Razporeditev odgovornosti 38
5.2 Cena in pogodba med plastmi 39
5.3 Konfiguracija in mesto omrežne povezave 39
5.4 Razumevanje uspeha in negativnega testa 40
5.5 Hevristični pregled 40
5.6 Prednostne izboljšave 42
5.7 Vrednotenje hipotez 42
5.8 Pomen skupnega zaledja 43
5.9 Vzdrževanje in odpravljanje napak 44
5.10 Veljavnost ugotovitev 44
5.11 Kaj bi spremenil v naslednjem preverjanju 45
5.12 Odgovor na primerjalno vprašanje 45
5.13 Razlaga finančnih zapisov 46
6 SKLEP 47
7 VIRI IN LITERATURA 49

KAZALO TABEL

Tabela 1 Raziskovalno gradivo 21

Tabela 2 Merila preverjanja 23

Tabela 3 Rezultati lokalnih zbirk 28

Tabela 4 Rezultati integracijskih scenarijev 32

Tabela 5 Rezultati dokumentiranega hevrističnega pregleda 41

Tabela 6 Vrednotenje hipotez 43

KAZALO SLIK IN GRAFIKONOV

Grafikon 1 Rezultati izvedenih lokalnih testnih zbirk 28

Slika 1 Izbira termina v aplikaciji React 33

Slika 2 Povzetek plačila v aplikaciji React s prikazom 122 evrov 34

Slika 3 Strežniško izračunan povzetek v aplikaciji Next.js 35

Slika 4 Prikaz potrditve dejansko plačane testne rezervacije 35

Slika 5 Dejanski mobilni prikaz izbire termina v Next.js 36

SEZNAM KRATIC IN IZRAZOV

API – programski vmesnik za povezovanje sistemov

CORS – pravila brskalnika za dostop do virov iz drugega izvora

HTTP in HTTPS – protokola za spletno komunikacijo; HTTPS vključuje zaščito prenosa

JSON – besedilni zapis strukturiranih podatkov

SDK – zbirka razvojnih pripomočkov in knjižnic

UI in UX – uporabniški vmesnik in uporabniška izkušnja

UTC – usklajeni univerzalni čas za primerjavo časovnih oznak

WCAG – smernice za dostopnost spletnih vsebin

PaymentIntent – objekt Stripe, s katerim spremljam življenjski cikel posameznega plačila

Webhook – strežniško obvestilo o dogodku v zunanjem sistemu

Idempotentnost – lastnost obdelave, pri kateri s ponovitvijo istega dogodka ne povzročim ponovitve poslovnega učinka

Staging – ločeno preizkusno okolje pred produkcijsko uporabo

1 UVOD

Spletna rezervacija mora hkrati izpolnjevati dve pričakovanji. Uporabnik želi izbrati storitev in termin, ponudnik pa želi vedeti, ali je rezervacija plačana in ali jo lahko izvede. Na zaslonu lahko tak postopek predstavim z nekaj gumbi, v ozadju pa povežem uporabniško sejo, ponudbo, razpoložljivost, plačilni nalog, zunanjega ponudnika plačil in podatkovno bazo. Če katero od teh povezav razumem napačno, lahko uporabniku prikažem napačno ceno ali potrditev, ki nima osnove v podatkih.

V diplomskem delu primerjam vključitev sistema Stripe v spletno rezervacijsko aplikacijo React s Supabase in v aplikacijo Next.js. Za praktični primer uporabim projekt Zeen Connect, v katerem imam dostop do obeh uporabniških izvedb in skupnega testnega zaledja. Primerjavo oprem na izvorno kodo, izvedene avtomatizirane preizkuse, dejanske testne transakcije Stripe ter pregled aplikacij v brskalniku. Poleg uspešnih poti namenoma preverim tudi zavrnitev plačila, preklic, napačen podpis dogodka in ponovno dostavo istega dogodka.

Za osrednjo enoto opazovanja izberem rezervacijo sintetične storitve QA Reiki Session 60. S tem ohranim enak poslovni primer pri pregledovanju obeh vmesnikov. Ne ocenjujem vsebine storitve ali njene učinkovitosti. Zanimajo me podatki, ki jih pri rezervaciji prenesem med sistemi, in način, kako uporabniku razložim stanje postopka. Za preizkuse uporabljam sintetičnega uporabnika in ponudnika, ločeno podatkovno bazo ter testni način Stripe.

1.1 Opredelitev problema

Najpomembnejši problem prepoznam v razliki med prikazom in potrjenim poslovnim stanjem. Besedilo, da je plačilo uspelo, lahko napišem na poljubno stran, vendar s tem še ne ustvarim rezervacije. Enako velja za ceno. Če znesek izračunam v brskalniku, lahko dobim drugačen rezultat kot na strežniku. Pri raziskavi zato ne preverjam le, ali se obrazec odpre, temveč tudi, ali njegov prikaz ustreza podatkom, ki jih zabeležim ob dejanski testni transakciji.

Drugi problem vidim v asinhroni obdelavi. Odgovor ponudnika plačil in obvestilo webhook ne prispeta nujno v trenutku, ko uporabnik pričakuje zaključek. Pri začasni napaki lahko prejmem isti dogodek večkrat. Če vsako dostavo obravnavam kot novo naročilo, ustvarim podvojene rezervacije ali finančne zapise. Če dogodek prezgodaj označim kot obdelan, lahko izgubim možnost pravilnega nadaljevanja. V praktičnem delu zato primerjam število poslovnih zapisov pred ponovitvijo dogodka in po njej.

Tretji problem povežem z razvojnim okoljem. Isti zahtevek lahko v strežniški funkciji deluje, v brskalniku pa zaradi pravil glede izvora zahtevka odpove. Takšno razliko moram prepoznati, preden jo pripišem kakovosti ogrodja. Pri preizkusu Reacta sem zato preveril tudi lokalni naslov aplikacije in uporabo dovoljene povezave do Supabase. Težavo z nastavitvijo okolja obravnavam ločeno od prikaza cen in uporabniških sporočil.

1.2 Namen in cilji

Z raziskavo želim ugotoviti, kako izbrana arhitektura vpliva na organizacijo plačilnega postopka in njegovo preverjanje. Pri Reactu spremljam neposredne klice iz brskalnika v Supabase Edge Functions. Pri Next.js spremljam strežniške akcije, ki iste zaledne funkcije vključijo v aplikacijski tok. Ne izhajam iz predpostavke, da eno ogrodje samo po sebi zagotavlja varen ali razumljiv rezultat. Takšno trditev preverjam prek konkretnih odgovornosti in opazovanih učinkov.

Kot prvi cilj določim pregled celotne povezave od storitve do plačila in rezervacije. Kot drugi cilj določim izvedbo ponovljivih preizkusov, pri katerih poleg odgovora API preverim tudi stanje v podatkovni bazi. Kot tretji cilj določim dokumentiranje dejanskega uporabniškega vmesnika s posnetki zaslona. Kot četrti cilj določim primerjavo ugotovitev, pri kateri ločim skupno zaledno logiko od razlik v uporabniških aplikacijah.

Pri pripravi rezultatov želim ohraniti sled od trditve do dokaza. Za vsak pomemben rezultat zato navedem uporabljeni scenarij in pričakovani učinek. Zaslonske slike uporabim za prikaz postavitve, besedila in zneskov; stanja Stripe in podatkovne baze preverim z ločenimi poizvedbami. Tako se pri sklepanju ne opiram samo na vtis, da aplikacija deluje, temveč na več medsebojno povezanih opažanj.

1.3 Raziskovalna vprašanja in hipoteze

Prvo raziskovalno vprašanje oblikujem okoli organizacije kode: Ali pri Next.js plačilno pot lažje povežem v aplikacijski strukturi kot pri ločenem vmesniku React in zaledju Supabase? Drugo vprašanje oblikujem okoli zanesljivosti: Ali pri izvedenih testih ohranim pravilno povezavo med plačilom in rezervacijo, tudi ko dogodek ponovim ali plačilo zavrnem? Tretje vprašanje oblikujem okoli uporabnosti: Ali na pregledanih zaslonih jasno prikažem ceno, izbrani termin, stanje postopka in možnost vrnitve?

Hipotezo H1 postavim tako: pri obravnavani izvedbi Next.js zmanjšam število uporabniških faz rezervacije in neposrednih zalednih povezav, ki jih upravljam v brskalniku. Hipotezo preverim s pregledom dejanskega toka in izvorne kode. Pri tem faze ne enačim s posameznim klikom. Ločeno zabeležim vidno razdelitev postopka in mesto, na katerem izvedem klic zaledja.

Hipotezo H2 postavim tako: s skupnim strežniškim preverjanjem cene, podpisa in že obdelanih dogodkov ohranim pravilno poslovno stanje pri izbranih pozitivnih in negativnih scenarijih. Pri preverjanju zahtevam več kot odgovor HTTP 200. Upoštevam, ali nastane rezervacija, ali je njen zapis povezan s plačilom in ali se ob ponovitvi dogodka število učinkov spremeni.

Hipotezo H3 postavim tako: pri obravnavanem primeru v Next.js dosežem preglednejši prikaz rezervacijskega postopka kot v primerjalni izvedbi React. Hipotezo preverim s strukturiranim hevrističnim pregledom in dejanskimi zasloni. Rezultat omejim na pregledani različici in izbrane naloge. Ne enačim ga z mnenjem vseh uporabnikov ali s splošno primerjavo obeh tehnologij.

1.4 Metode in razmejitev raziskave

Uporabim študijo primera, primerjalni pregled kode, avtomatizirano testiranje in strokovni pregled vmesnika. Teste izvedem 16. septembra 2026. Pri lokalnih zbirkah shranim celotne izpise. Pri integracijskih preizkusih shranim strukturirane rezultate z datumom, identifikatorjem zagona, preverjenimi stanji in odtisi identifikatorjev. Pri pregledovanju vmesnika shranim dejanske zaslonske slike obeh aplikacij.

Plačilne scenarije izvajam v testnem načinu. Pri opisu zato dosledno razlikujem med dejanskim klicem testnega API-ja Stripe in prenosom resničnega denarja. Preizkus s testno kartico mi omogoči preverjanje integracije, ne pomeni pa nakupa storitve pri dejanski stranki. Prav tako ne izvajam raziskave z uporabniki, zato pri hevrističnem pregledu govorim o lastnih ugotovitvah, ne o izmerjenem zadovoljstvu udeležencev.

Kot mejo raziskave določim funkcionalni rezervacijski in plačilni tok. Ne izvajam obremenitvenega napada, celovitega varnostnega pregleda ali revizije davčne zakonodaje. Davčne zneske obravnavam kot rezultate konfiguriranega testnega primera. S tem ohranim jasno raziskovalno vprašanje: ali obe uporabniški izvedbi pravilno vključita plačilni postopek in ali lahko njegove rezultate preverim z ohranjenimi dokazi.

1.5 Zgradba dela

V drugem poglavju razložim pojme, s katerimi opisujem plačilo, rezervacijo in obdelavo dogodkov. V tretjem poglavju določim okolje, merila in postopek preizkušanja. V četrtem poglavju predstavim izvedene teste in pregled uporabniških poti. V petem poglavju primerjam ugotovitve ter ovrednotim hipoteze. V sklepu odgovorim na raziskovalna vprašanja in izpostavim spremembe, ki jih na osnovi rezultatov štejem za najpomembnejše. V elektronskih prilogah ohranim rezultate in slike, da lahko svoje ugotovitve preverim tudi po koncu pisanja.

2 TEORETIČNA IZHODIŠČA

Pri plačilni integraciji razlikujem med uporabniškim vmesnikom, aplikacijskimi pravili, trajnim zapisom in zunanjim plačilnim sistemom. Posamezno napako lahko pravilno razložim šele, ko vem, katera od teh plasti je odgovorna za določen podatek. V tem poglavju zato najprej pojasnim stanja, nato povezave med tehnologijami in nazadnje merila, s katerimi ocenim zanesljivost ter uporabnost.

2.1 Rezervacija in plačilo

Rezervacijo obravnavam kot dogovor o izbrani storitvi in času izvedbe. Plačilo obravnavam kot ločen postopek poravnave. Med njima vzpostavim povezavo, vendar njunih stanj ne združim v eno samo oznako. Pri isti rezervaciji lahko hkrati zabeležim uspešno plačilo in čakanje na izvajalčevo potrditev termina. Takšno razlikovanje potrebujem tudi pri odpovedi, spremembi termina in vračilu.

Če bi za vse uporabil le oznaki uspeh in napaka, bi izgubil pomemben del poslovnega pomena. Pri plačilu, ki je že sprejeto, vendar še nima dokončnega računa, ne želim uporabniku predlagati novega plačila. Pri zavrnjeni kartici pa mu ne želim prikazati rezervacije kot plačane. Zato v modelu ločim stanje plačilnega naloga, stanje plačila rezervacije in stanje nadaljnje finančne obdelave. Pri branju rezultatov nato preverim kombinacijo teh vrednosti.

Povezavo med sistemi oblikujem z identifikatorji. Notranji identifikator rezervacije uporabim za poslovne operacije, identifikator plačila Stripe za komunikacijo s ponudnikom plačil, identifikator dogodka pa za evidenco njegove obdelave. Nobenega od njih ne uporabim kot nadomestilo za preverjanje pravic. Če poznam identifikator tujega zapisa, s tem še ne smem pridobiti možnosti za njegovo spreminjanje.

2.2 Checkout, PaymentIntent in vdelani obrazec

Pri pregledu gradiva razlikujem med začetnim poenostavljenim prototipom s Checkout Session in dejansko aplikacijo s PaymentIntent ter vdelanim plačilnim obrazcem. Checkout Session mi omogoča organizacijo seje nakupa, PaymentIntent pa predstavlja življenjski cikel posameznega plačila. V praktičnem delu uporabljam slednjega, zato dogodkov payment_intent.succeeded in payment_intent.canceled ne poimenujem kot dogodke zaključka seje Checkout (Stripe, b. d.-a; Stripe, b. d.-b)

To razlikovanje je zame pomembno pri izbiri testnih pričakovanj. Pri enem načinu lahko uporabnika preusmerim na gostovano stran, pri drugem pa v svojo postavitev vključim element Stripe. V obeh primerih moram strežniško določiti, kaj uporabnik plačuje, ter po plačilu preveriti rezultat. Izbira obrazca sama po sebi ne odgovori na vprašanje, ali sem pravilno povezal plačilo z rezervacijo.

Pri vdelanem obrazcu kartične podatke oddam prek elementa ponudnika plačil. V svoji aplikaciji obdelujem identifikatorje in rezultat postopka. Za programske preizkuse uporabim pripravljena testna plačilna sredstva, na primer pm_card_visa in pm_card_chargeDeclined. S tem v testno kodo ne vključim resničnih kartičnih podatkov. Testni način uporabim za preverjanje dogodkov, ki bi jih bilo z dejanskim denarjem neprimerno namenoma povzročati, na primer zavrnitev ali spor (Stripe, b. d.-e).

2.3 Cena kot strežniška odločitev

Pri ceni razlikujem med prikazom in avtoritativnim izračunom. Vmesnik lahko znesek oblikuje za prikaz uporabniku, dokončno odločitev pa oprem na osnovi ponudbe in pravil v zaledju. Če uporabnik pošlje drug znesek, pričakujem, da ga strežnik ne sprejme kot novo ceno storitve. To lastnost v praktičnem delu preverim z namerno spremenjenim podatkom amount.

V svojih testih zneske primerjam v najmanjši denarni enoti. Pri evru tako primerjam cente, ne pa oblikovanih nizov z vejico, piko ali znakom valute. Znesek 100 evrov zapišem kot 10000. Takšna primerjava mi omogoči nedvoumno preverjanje vsote neto in davčnega dela. Pri prikazu uporabniku pa ohranim oblikovanje, ki ustreza jeziku in valuti.

Posebej ločim ceno, ki že vključuje davek, od osnove, ki ji davek šele prištejem. Če znesek 100 evrov obravnavam kot bruto ceno, pri 22-odstotni stopnji dobim osnovo 81,97 evra in davčni del 18,03 evra. Če isti znesek obravnavam kot osnovo, prikažem 122 evrov. Razliko lahko povzročim brez napake v osnovnem množenju; napačno sem določil pomen vhodnega podatka. Prav to razliko zato preverjam kot vprašanje pogodbe med vmesnikom in zaledjem.

Pri presoji davčnega prikaza ne podajam splošnih davčnih nasvetov. Spremljam konfigurirani sintetični primer in skladnost izračuna med plastmi aplikacije. Za tehnično pravilnost zahtevam, da razumem pomen vsakega zneska, da ohranim valuto in da razčlenitev uporabniku pokaže isti skupni znesek, kot ga uporabim pri plačilu. Pravna pravilnost davčnih pravil ostaja ločena strokovna naloga.

2.4 Webhook in pristnost dogodka

Webhook uporabim kot strežniško pot za prejem dogodka iz sistema Stripe. Pri tem ne zaupam samo imenu dogodka ali vsebini JSON. Preverim podpis nad nespremenjenim telesom zahtevka in upoštevam skrivnost ustrezne končne točke. Če bi telo pred preverjanjem sestavil drugače, bi lahko spremenil podatke, na katerih temelji podpis. Pri izvedbi zato preverjanje umestim pred poslovno obdelavo (Stripe, b. d.-c; Stripe, b. d.-d).

Pri testiranju ločim tri vprašanja. Najprej preverim, ali sprejmem pravilno podpisan zahtevek. Nato preverim, ali zavrnem zahtevek z manjkajočim ali napačnim podpisom. Nazadnje preverim, ali znan tip dogodka povzroči pravilen poslovni učinek. Pravilno podpisan neznan dogodek lahko varno prezrem, zato odgovora HTTP 200 pri takem dogodku ne enačim s potrjeno rezervacijo.

V testnem okolju dodatno preverim način dogodka. Dogodka z oznako livemode true ne želim obdelati v testnem poslovnem toku. V negativnem preizkusu takšno oznako vključim v sintetični podpisani dogodek in preverim odziv testne končne točke. S tem ne ustvarim produkcijskega plačila, temveč preverim mejo, s katero preprečujem mešanje načinov.

2.5 Ponovitve, sočasnost in idempotentnost

Idempotentnost razumem kot lastnost, pri kateri ponovljena obdelava istega dogodka ne povzroči dodatnega poslovnega učinka. Pri plačilu me zato ne zanima samo, ali na ponovitev vrnem uspešen odgovor. Preverim tudi število rezervacij, transakcij, zapisov v knjigi finančnih dogodkov in obdelanih webhook dogodkov. Če se katera od teh količin nepričakovano poveča, ponovitev ni ostala brez poslovne posledice.

Zaporedno in sočasno ponavljanje obravnavam ločeno. Pri zaporednem primeru lahko prvi zahtevek že zapiše rezultat, preden prejmem drugega. Pri sočasnem primeru pa lahko več izvajanj hkrati prebere začetno stanje. Zato zgolj preverjanje, ali identifikator že obstaja, ni vedno dovolj. Potrebujem še usklajeno trajno obdelavo, pri kateri isto spremembo izvede samo eno izvajanje.

Pri lokalnem prototipu lahko takšno obnašanje ponazorim s pomnilniško množico. Pri gostovanem zaledju pa potrebujem dokaz, ki preživi posamezen proces in temelji na podatkovnih zapisih. V raziskavi zato izvedem tri zaporedne in deset sočasnih ponovitev dejanskega testnega dogodka. Števila izberem kot funkcionalni preizkus podvajanja, ne kot meritev največje zmogljivosti sistema.

Pri obravnavi ponovitev upoštevam še razliko med dostavo in učinkom. Več dostav je lahko pričakovano vedenje komunikacijskega sistema. Več ustvarjenih rezervacij za eno plačilo pa bi pomenilo poslovno napako. Zasnovo zato presojam po ohranjenem rezultatu, ne po zahtevi, da bi se dogodek pojavil samo enkrat.

2.6 React in neposredna povezava s Supabase

React uporabim za sestavljanje vmesnika iz komponent. V obravnavani aplikaciji rezervacijo vključim v modalno okno, v katerem spreminjam korake in ohranjam izbrani termin. Pri pregledu spremljam, katere podatke hranim v stanju komponente ter katere pridobim z zalednim klicem. Opiram se na ločitev prikaza in interakcij, kot jo opisuje dokumentacija React (React, b. d.).

V pregledani izvedbi iz brskalnika pošljem zahtevo v Supabase. Zato moram poleg uporabniške seje upoštevati tudi pravila dostopa med izvori. Lokalni naslov aplikacije postane del konfiguracije preizkusa. Če izberem vrata, ki niso dovoljena, lahko dobim napako omrežnega dostopa, čeprav sama zaledna funkcija deluje. Pri primerjavi takšen primer uporabim za razlago dodatne povezave, ki jo moram upravljati.

Supabase obravnavam kot skupno mesto za avtentikacijo, podatke in zaledne operacije. Pri rezervaciji mi to omogoči, da različna vmesnika uporabljata ista pravila. Varnostne omejitve zato ne ostanejo samo v gumbu ali obrazcu. Pri tajnih podatkih uporabim strežniško okolje in jih ne vključim v gradivo, ki ga prenesem v brskalnik (Supabase, b. d.-a; Supabase, b. d.-b).

2.7 Next.js in strežniške akcije

Pri Next.js ločim komponente, ki izvajajo uporabniške interakcije, od strežniških operacij. V dejanskem primeru plačilno pot spremljam prek strežniških akcij v datoteki booking-actions.ts. Te akcije preverijo vhodne podatke in vključijo funkcije Supabase. Zato v primerjavi ne trdim, da sem celotno plačilno zaledje prenesel v Next.js; premaknem predvsem mesto, s katerega uporabniški tok pokliče skupno zaledje.

Route Handler razumem kot drugo možnost organizacije strežniškega zahtevka, vendar ga ne enačim s strežniško akcijo. V začetnem prototipu sem pregledal takšno pot, v razširjenem praktičnem primeru pa opazujem dejansko uporabljene akcije. S tem poimenovanje v diplomskem delu uskladim s kodo, ki sem jo preizkusil (Next.js, b. d.-b; Next.js, b. d.-c).

Pri okoljskih spremenljivkah preverim ločitev javnih vrednosti in strežniških skrivnosti. Predpono NEXT_PUBLIC_ uporabim samo za vrednosti, ki jih smem vključiti v uporabniški del. Skrivnega ključa Stripe ne potrebujem za izris povzetka rezervacije, zato ga ne posredujem komponenti. Pri raziskovalnem paketu izključim tudi datoteke z dostopnimi podatki testnih uporabnikov (Next.js, b. d.-a).

2.8 Preverjanje na različnih ravneh

Pri testiranju ločim lokalne teste, integracijske scenarije in pregled v brskalniku. Z lokalnim testom hitro preverim posamezno pravilo ali pričakovano strukturo kode. Z integracijskim scenarijem preverim povezavo več sistemov. V brskalniku pa preverim, ali uporabnik vidi razumljiv rezultat in ali lahko izvede predvideno dejanje. Nobene od teh ravni ne uporabim kot popoln nadomestek za drugi dve.

Takšno razdelitev uporabim tudi pri štetju. Število uspešnih lokalnih testov navedem kot velikost izvedene zbirke. Iz njega ne izračunam odstotka preverjene kode, ker za to potrebujem posebno meritev pokritosti. Prav tako sedmih integracijskih scenarijev ne prištejem k stotinam lokalnih testov in iz vsote ne izpeljem skupne ocene varnosti. Vsaki številki pripišem njen dejanski pomen.

Pri časovnih meritvah upoštevam vzorec in vrsto zahtevka. Pet kratkih odgovorov končne točke mi pomaga prepoznati nepričakovano zakasnitev, vendar mi ne omogoča zanesljive napovedi vedenja pri velikem številu uporabnikov. V rezultatih zato navedem izmerjene vrednosti in velikost vzorca ter se izognem trditvam o produkcijski zmogljivosti, ki jih tak preizkus ne podpira.

2.9 Uporabnost in dostopnost

Za hevristični pregled uporabim Nielsenov okvir desetih načel. Preverim vidnost stanja, razumljivost izrazov, možnost vrnitve, doslednost, preprečevanje napak, prepoznavanje informacij, učinkovitost, preglednost, pomoč pri napaki in dosegljivost pojasnil. Pri vsakem načelu zabeležim konkretno dejanje in opaženi rezultat. Tako svojo presojo vežem na zaslon, ne na splošen vtis o tehnologiji (Nielsen, 1994).

Dostopnost obravnavam kot povezano, vendar ločeno vprašanje. Z oznakami, semantičnimi elementi in jasnim fokusom lahko uporabniku olajšam delo tudi brez miške. Pri pregledu preverim prisotnost oznak in vlogo elementov, vendar iz tega ne izpeljem potrditve celotne

skladnosti z WCAG 2.2. Za takšno presojo bi moral uporabiti širši in posebej opredeljen postopek (World Wide Web Consortium [W3C], 2024).

Pri končni oceni uporabnosti dam največjo težo pravilnosti poslovnih informacij. Estetsko urejeno okno z napačno ceno zame ni uspešno rešena plačilna pot. Prav tako ne štejem za dovolj dobro obvestilo, ki ob omrežni napaki trdi, da ponudnik nima urnika. Pri pregledovanju zato vsebino sporočila vedno primerjam z vzrokom, ki sem ga ugotovil v tehničnem delu.

2.10 Lastništvo stanja in obnova postopka

Pri razdelitvi odgovornosti za vsak podatek določim mesto, na katerem ga štejem za potrjenega. Izbrano uro pred oddajo lahko hranim v stanju obrazca. Razpoložljivost preverim v zaledju, uspeh plačila pri ponudniku plačil, obstoj rezervacije pa v aplikacijski bazi. Če ta mesta zamenjam, lahko začasno izbiro v brskalniku napačno obravnavam kot že zaključeno poslovno odločitev.

Takšno razmišljanje uporabim tudi pri ponovni naložitvi strani. Uporabnik lahko osveži brskalnik, odpre drugo napravo ali zapre okno med plačilom. Pri obnovi zato ne želim biti odvisen samo od spremenljivke, ki je obstajala v prejšnjem prikazu. Če imam trajni plačilni nalog in ustrezno preverjanje pravic, lahko stanje znova preberem ter sestavim pravilno sporočilo. Statusno stran v praktičnem delu uporabim prav kot primer takšnega branja.

Pri čakanju na zaledno obdelavo določim tudi mejo samodejnega osveževanja. Brez nje lahko uporabniku neomejeno prikazujem nalaganje, čeprav potrebujem drugačen naslednji korak. Po drugi strani lahko s prekratkim čakanjem uspešno, vendar nekoliko počasnejšo obdelavo prikažem kot napako. Zato bi časovni potek povezoval z znanim stanjem: ali čakam na plačilo, na dogodek, na nastanek rezervacije ali na poznejšo finančno obdelavo.

Pri ponovnem poskusu preverim, kaj sploh ponavljam. Ponovno branje statusa je drugačno dejanje od ustvarjanja novega plačila. Če je obstoječe plačilo uspešno, uporabnika ne želim vrniti na začetno bremenitev. Če je njegov nalog preklican, mu lahko ponudim nov postopek z novo izbiro. S tem omejim možnost, da prijazno oblikovan gumb za ponovitev skrije napačno poslovno operacijo.

Pri načrtovanju sporočil iz takšne razdelitve izpeljem jasnejše besedilo. Pri izbiri termina govorim o izboru, pri oddaji o preverjanju, po potrjenem zapisu o rezervaciji in pri vračilu o stanju vračila. Enakih besed ne uporabljam za različne poslovne dogodke samo zato, ker se

zgodijo v istem oknu. Tako zmanjšam potrebo po tem, da bi uporabnik poznal notranje tehnične oznake.

3 PRIMERJALNA ZASNOVA RAZISKAVE

Raziskavo zasnujem kot preverjanje dveh uporabniških izvedb istega poslovnega primera. Pri tem skupno zaledje uporabim kot kontrolirano povezavo, s katero zmanjšam razlike v ponudbi, podatkih in plačilnih pravilih. Razlike med aplikacijama iščem predvsem v organizaciji klicev, prikazu cene, korakih postopka in sporočilih. Zaledne preizkuse izvedem ločeno, da uspeha iste funkcije ne pripišem dvakrat različnima ogrodjema.

3.1 Gradivo in sledljivost različic

Najprej pregledam začetna prototipna sklopa, ki sta priložena diplomskemu delu. V njiju prepoznam poenostavljena pravila za spremembo stanja in lokalne teste. Za razširjeni praktični del nato uporabim dejanski projekt Zeen Connect z aplikacijo React, podprojektom Next.js, funkcijami Supabase in pripravljenimi testnimi skriptami. Izvorno gradivo ohranim ločeno od rezultatov novega preizkusa, da ne pomešam starega modela in nove meritve.

Pri pregledu kode izberem datoteke, skozi katere lahko sledim rezervacijskemu toku. V Reactu spremljam BookingModal.jsx, povezavo Supabase in pomožne funkcije za obdelavo stanja rezervacije. V Next.js spremljam booking-flow.tsx, booking-actions.ts in booking-status-watcher.tsx. Pri zaledju spremljam končni točki za ustvarjanje plačila in obdelavo dogodkov. S tem določim omejen, vendar vsebinsko povezan vzorec kode.

Za ponovljivost ohranim izpise testov, strukturirane rezultate in posnetke zaslona. Pri datotekah v oddajnem paketu pripravim popis z zgoščevalnimi vrednostmi. S takšnim popisom lahko pozneje preverim, ali je posamezna dokazna datoteka ostala enaka. Zgoščevalna vrednost mi ne dokazuje pravilnosti vsebine; uporabim jo kot pripomoček za preverjanje njene nespremenjenosti.

Tabela 1 Raziskovalno gradivo

Sklop	**Kaj pregledam**	**Dokaz**
React	Rezervacijsko okno in neposredne klice Supabase	BookingModal.jsx in slike
Next.js	Rezervacijsko stran, akcije in prikaz statusa	booking-flow.tsx in povezane datoteke
Skupno zaledje	Plačila, dogodke in poslovne zapise	Rezultati Stripe in Supabase
Lokalni testi	Tri projektne zbirke	466 + 307 + 175 preverjanj
Vmesnik	Obe aplikaciji v brskalniku	Zaslonske slike in zapisnik

Vir: moj pregled in izvedeni preizkusi 16. 9. 2026

3.2 Testno okolje

Za podatkovno zaledje uporabim projekt zeen-connect-staging. Pred transakcijami preverim, da povezava ne kaže na produkcijski projekt, da je način Stripe nastavljen na test in da imam ločeni skrivnosti za plačilno končno točko ter končno točko Connect. Nato z API-jem Stripe pregledam registrirane webhook povezave. V svežem popisu preverim osem povezav in deset povezanih testnih računov.

Pri pregledu povezav ugotovim, da sta ustrezni plačilna povezava in povezava Connect za staging omogočeni, produkcijska plačilna povezava v testnem računu pa je onemogočena. Ta podatek uporabim za preverjanje usmerjanja dogodkov pred testom. Ne sklepam samo iz imena datoteke ali starega navodila, saj lahko konfiguracijo med razvojem spremenim. Za dokaz zato ohranim dejanski popis, zajet ob izvedbi.

Uporabim sintetičnega odjemalca Zeen QA Client Ana in sintetičnega ponudnika Zeen QA Provider Ljubljana. V scenarijih storitev prepoznam po naslovu QA Reiki Session 60 in preverim njeno ceno ter trajanje. Testni računi mi omogočijo preverjanje pravic uporabnika in ponudnika brez poseganja v poslovanje dejanskih strank. Dostopnih podatkov ne vključim v diplomsko delo, izpise ali arhiv za oddajo.

Uporabniški aplikaciji za primerjavo zaženem lokalno in ju povežem z istim testnim zaledjem. React pregledam na naslovu 127.0.0.1 z vrati 3000, Next.js pa na vratih 3100. Pred tem preverim tudi oddaljeno staging stran in prvi lokalni zagon Reacta na drugih vratih. Za končne primerjalne posnetke izberem delujoči nastavitvi; začetni težavi ohranim kot zapis o pripravi okolja, ne kot dokaz, da ogrodje ne podpira rezervacij.

3.3 Enak poslovni primer

Pri obeh vmesnikih izberem isto storitev, istega uporabnika in isti časovni pas Europe/Ljubljana. Zaradi zasedenosti in začasnih zadržanj pri posameznih preizkusih ne vztrajam pri popolnoma isti uri. Namesto tega preverim isti postopek izbire prostega termina. Takšno razliko posebej upoštevam pri primerjavi zaslonskih slik: na sliki lahko vidim drugo uro, še vedno pa opazujem isto vrsto storitve in ista pravila plačila.

Pri integracijskih skriptah začasno nastavim sintetični davčni kontekst, ki ga po koncu obnovim. Pri razširjenem scenariju uporabim ponudnika, ki v testnem primeru ni davčni zavezanec, zato osnovno plačilo znaša 100 evrov brez davčnega dela na storitvi. Pri poznejšem

uporabniškem pregledu opazujem obnovljeni profil z 22-odstotno nastavitvijo. Razlike med tema primeroma ne obravnavam kot napako v podatkih; navedem jo kot del vhodnih pogojev.

Za primerjavo uporabniških cen zato uporabim zaslona, zajeta po obnovi nastavitev. Takrat v Reactu preberem 122 evrov, v Next.js pa skupni znesek 100 evrov z razčlenitvijo na 81,97 in 18,03 evra. Ker v obeh primerih opazujem isto katalogizirano storitev, lahko razliko razložim z različnim razumevanjem osnovne cene. Test z manipuliranim vhodnim zneskom mi dodatno pokaže, da končno ceno določi zaledje.

3.4 Merila uspeha

Za uspešno plačilo zahtevam potrjen rezultat v sistemu Stripe in povezan zapis rezervacije. Poleg tega preverim finančni transakciji, davčni posnetek in zapise v finančni knjigi. Za zavrnjeno plačilo zahtevam stanje neuspeha in odsotnost ustvarjene rezervacije. Za preklic zahtevam stanje canceled v sistemu Stripe ter ustrezno stanje cancelled pri internem nalogu. Razliko v zapisu imen obravnavam kot preslikavo med sistemoma.

Pri ponovljenem dogodku zahtevam nespremenjeno število poslovnih učinkov. Pri vračilu zahtevam dejanski testni objekt vračila Stripe s pričakovanim zneskom in rezultat odobritve v aplikaciji. Pri neavtorizirani zahtevi zahtevam zavrnitev. Pri napačnem podpisu zahtevam zavrnitev pred poslovno obdelavo. Takšna pričakovanja določim pred razlago rezultatov, da uspeha ne prilagodim šele opaženemu odgovoru.

Pri pregledu vmesnika uporabim drugačna merila. Preverim, ali lahko izberem termin, ali pred nadaljevanjem vidim povzetek, ali je gumb brez potrebnega vnosa onemogočen, ali se lahko vrnem in ali ob potrditvi vidim razumljivo sporočilo. Če pri posameznem dejanju opazim napako, test vseeno štejem kot izveden. V rezultat zapišem dejansko opažanje in ga ne preimenujem v uspeh samo zato, ker je zaledni preizkus uspel.

Tabela 2 Merila preverjanja

Merilo	**Kaj preverim**	**Pričakovani rezultat**
Uspešno plačilo	Stripe in povezano rezervacijo	Skladna plačana rezervacija
Neuspeh ali preklic	Stanje naloga in booking_id	Brez nove rezervacije
Ponovitev dogodka	Količine pred ponovitvijo in po njej	Brez dodatnih učinkov
Podpis	Manjkajoč, napačen in veljaven podpis	Ustrezna zavrnitev ali sprejem
Cena	Vhodni znesek in strežniški odgovor	Katalogizirana cena
Uporabniški prikaz	Korake, povzetek, napake in potrditev	Razumljivi in pravilni podatki

Vir: moj pregled in izvedeni preizkusi 16. 9. 2026

3.5 Zaporedje izvedbe

Najprej izvedem popis okolja in preverim zaščitne pogoje testnih skript. Nato izvedem osnovni plačilni scenarij, s katerim preverim, ali povezava med sistemom Stripe in staging zaledjem deluje. Šele po uspešnem preizkusu nadaljujem z razširjenim scenarijem, ki vključuje ponovitve dogodkov, spremembo termina, vračilo, zavrnitev plačila, spor in testno izplačilo Connect. S tem lažje ločim težavo v osnovni povezavi od napake posameznega nadaljnjega postopka.

Ločeno izvedem preizkus meja končnih točk webhook. Pri vsaki uporabim pet pravilno podpisanih neobravnavanih dogodkov in po en dogodek brez podpisa, z napačnim podpisom ter z neustrezno oznako načina. Takšna razdelitev mi omogoči, da natančno povem, kateri del podpisovanja sem preveril. Sintetičnega neobravnavanega dogodka ne predstavim kot pravi dogodek uspešnega plačila.

Nato izvedem dodatni scenarij za avtoritativnost cene, preklic in zahtevo brez prijave. Pri ceni pošljem vhodno vrednost ena ter primerjam odgovor s katalogizirano ceno storitve. Ustvarjeni plačilni nalog nato prekličem v sistemu Stripe in počakam na rezultat v zaledju. Tako v istem preizkusu preverim tudi, ali se preklic prenese prek dejanske povezave webhook.

Lokalne zbirke testov izvedem za osnovni projekt in za oba tipa testnih datotek v Next.js. Rezultat vsakega zagona shranim ločeno. Pri pregledu v brskalniku začnem s prijavo, nadaljujem z izbiro storitve ter termina in preverim pot do plačilnega koraka. Potrditev rezervacije dodatno preverim z odprtjem dejanskega statusa že izvedene testne transakcije. Pri tem ne spreminjam stanja prek naslovne vrstice; stran le prebere stanje iz zaledja.

3.6 Zajem in razvrščanje dokazov

Dokaze razvrstim po namenu. V izpisih lokalnih testov ohranim imena preverjanj in končno število uspehov. V datotekah JSON ohranim rezultate integracijskih scenarijev in primerjave zapisov. Na zaslonskih slikah ohranim dejanski prikaz aplikacije. V popisu kode ohranim izbrane datoteke, s katerimi razložim opažene razlike. Takšna razvrstitev mi olajša povezovanje med besedilom diplomskega dela in priloženim gradivom.

Pri identifikatorjih uporabim kratke zgoščene odtise, kadar za razumevanje rezultata ne potrebujem celotnega identifikatorja. S tem lahko povežem ponovitve istega scenarija, ne da bi v besedilu navajal obsežne tehnične reference računov. Skrivne ključe, žetone seje, gesla in vsebino konfiguracijskih datotek iz paketa izključim. Za ponovitev testov navedem potrebne vrste nastavitev, ne njihovih dejanskih vrednosti.

Posnetkov aplikacije ne ustvarim z risanjem nadomestnih zaslonov. Zajamem jih v brskalniku med preizkusom. Zato lahko na njih ostanejo tudi razvojna oznaka Next.js, sintetična imena in negativni rezultati, ki sem jih opazil. Te značilnosti v razlagi navedem, ker bralcu pomagajo razumeti okolje. Za dokaz delovanja raje uporabim resničen zaslon z jasno razlago kot vizualno olepšan prikaz, ki ne predstavlja izvedenega testa.

3.7 Omejitve sklepanja

Pri primerjavi upoštevam, da aplikaciji nista dve neodvisno in naključno izbrani rešitvi. Obravnavam dve izvedbi istega projekta, pri katerih uporabljam skupno zaledje in del poslovnih pravil. Ugotovitve zato povežem z njuno zasnovo in zrelostjo. Iz njih ne izpeljem trditve, da je vsaka aplikacija Next.js boljša od vsake aplikacije React, saj bi za tak sklep potreboval drugačno raziskavo.

Pri uporabnosti nastopam kot en ocenjevalec. Ne izvajam intervjujev, meritev časa več udeležencev ali statistične primerjave uspešnosti nalog. Namesto takšnih podatkov pripravim dokumentiran strokovni pregled. Kadar napišem, da je sporočilo premalo jasno, navedem njegovo vsebino in razlog za presojo. S tem ohranim možnost, da bralec oceno preveri na istem primeru.

Pri obravnavi napak upoštevam tudi brskalnik in način zagona. Prazen kartični okvir v pregledanem okolju zabeležim kot rezultat preizkusa nalaganja. Ne posplošim ga na vse brskalnike, vendar ga ne izključim zato, ker je plačilni API uspešno izvedel transakcijo. Prav povezava med uspešnim zalednim testom in neuspešnim uporabniškim korakom mi pokaže, zakaj pri preverjanju potrebujem več ravni dokazov.

3.8 Natančnost zapisnika in preverjanje nasprotujočih si podatkov

Pri sestavljanju zapisnika ohranim čas začetka in zaključka posameznega integracijskega zagona. Za razširjeni scenarij sem shranil začetek 16. septembra 2026 ob 15.17.49 UTC in zaključek ob 15.19.01 UTC. Te vrednosti uporabim za povezovanje dokazov, ne kot meritev trajanja posameznega uporabniškega plačila. V skupnem času namreč izvedem več zaporednih operacij in čakam na različne dogodke.

Časovne oznake v dokazih primerjam v istem časovnem standardu. Pri zaslonih pa upoštevam časovni pas Europe/Ljubljana. Tako ne zamenjam časa, ob katerem sem izvedel test, s terminom, ki ga je uporabnik rezerviral, ali z lokalnim časom prikaza. Pri rezervacijskem

sistemu je ta razlika pomembna, ker ista ura na dveh napravah brez navedbe časovnega pasu še ne pomeni istega trenutka.

Ko v dveh dokazih najdem različen podatek, najprej preverim njun pomen. Pri ceni preverim davčni kontekst in to, ali berem neto ali bruto znesek. Pri stanju preverim, ali opazujem plačilo, rezervacijo ali finančni zaključek. Pri številu zapisov preverim, ali je med zajemoma potekel dodaten scenarij. S tem preprečim, da bi vsako razliko samodejno označil kot programsko napako.

Tak pristop sem uporabil pri stanju processing. Najprej sem ga opazil v rezultatu uspešnega plačila, nato sem preveril še naknadni zapis in kodo prikaza. Šele nato sem pojasnil, da v obravnavani aplikaciji to stanje lahko spremlja že plačano rezervacijo. Pri ceni pa sem po preverjanju konteksta ohranil ugotovitev o neskladju, ker sta uporabniška prikaza različno obravnavala katalogizirani znesek.

Pri končnem izboru dokazov ohranim tudi negativna opažanja. Če bi v paket vključil samo posnetke, na katerih je vse videti uspešno, bi izgubil pomemben del raziskave. Zato ob potrditvenem zaslonu ohranim še cenovna povzetka in zapis o praznem kartičnem okvirju. S tem pripravim gradivo, ki podpira dejansko razpravo o kakovosti in ni samo predstavitev želenega rezultata.

4 PRAKTIČNI PREGLED IN PREIZKUS

V praktičnem delu preverim povezavo med uporabniško aplikacijo, funkcijami Supabase in testnim računom Stripe. Rezultate predstavim v zaporedju, ki mi omogoča razlikovanje med pripravo okolja, poslovnimi učinki in uporabniškim prikazom. Pri vsakem scenariju navedem, kaj sem sprožil, kaj sem pričakoval in kaj sem dejansko preveril. Uspešnih odgovorov ne uporabim kot nadomestilo za pregled ustvarjenih zapisov.

4.1 Priprava in prvi zagon

Pred prvim plačilom sem izvedel popis Stripe sandboxa. Preveril sem povezane končne točke in način njihovega delovanja. V popisu sem ohranil naslove, omogočenost ter tipe dogodkov. Pri plačilni povezavi staging sem preveril obravnavo uspeha, neuspeha in preklica PaymentIntent. Pri povezavi Connect sem preveril dogodke povezanega računa in izplačil. Za izvedbo sem uporabil že pripravljeno testno okolje projekta.

Pri prvem osnovnem scenariju sem se prijavil s sintetičnim uporabnikom, poiskal testno storitev in izbral prost termin. Skripta je preverila tudi stanje sintetičnega ponudnika. Pred plačilom sem zahteval testni način in veljavne identifikatorje plačilnega naloga. Nato sem v sistemu Stripe potrdil plačilo s pripravljenim plačilnim sredstvom pm_card_visa. S tem sem izvedel dejanski klic testnega API-ja in sprožil testne dogodke Stripe.

Po potrditvi nisem takoj zaključil preverjanja. Počakal sem, da je webhook ustvaril oziroma povezal rezervacijo, ter preveril finančne zapise. Osnovni scenarij je uspel. Ta rezultat sem uporabil kot izhodišče za razširjene scenarije, saj sem z njim preveril, da je osnovna pot med testnim plačilom in staging bazo pravilno povezana.

4.2 Lokalni testi obeh projektov

V osnovnem projektu sem izvedel zbirko 466 lokalnih testov. V podprojektu Next.js sem ločeno izvedel 307 testov v datotekah MJS in 175 testov v datotekah TypeScript. V vseh treh zagonih nisem imel neuspešnih, preklicanih ali preskočenih testov. Skupaj sem tako izvedel 948 lokalnih preverjanj. Izpise sem shranil v datoteke react-suite.txt, next-suite-mjs.txt in next-suite-ts.txt.

Pri razlagi te številke upoštevam vsebino zbirk. V osnovnem projektu ne preverjam samo plačilnega obrazca, temveč tudi druga poslovna pravila in zaščitne pogoje projekta. V Next.js

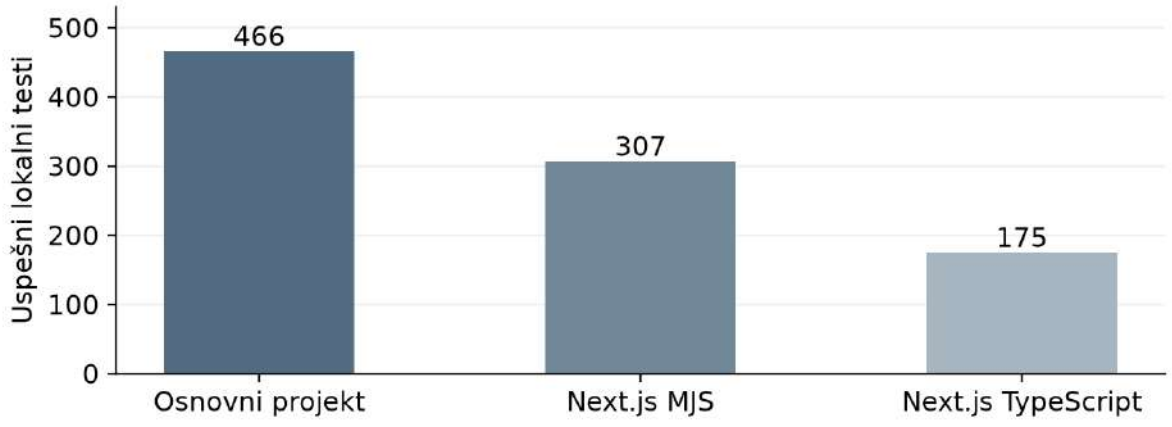
prav tako vključim širši nabor pravil ter preverjanj strukture. Zato rezultat 948 uspešnih testov predstavim kot uspeh izvedenih projektnih zbirk. Ne napišem, da sem opravil 948 neodvisnih testnih plačil ali da sem s tem preveril vso izvorno kodo.

Lokalni testi so mi pomagali preveriti, ali pregledano izhodišče izpolnjuje pripravljena avtomatizirana pričakovanja. Ko sem pozneje v brskalniku ugotovil razliko v ceni, rezultatov lokalnih testov nisem spreminjal. Namesto tega sem zabeležil razliko med tem, kar preverjajo testne zbirke, in tem, kar sem opazil v uporabniškem toku. Takšen primer mi pokaže, zakaj uspešen zagon zbirke še ne pomeni zaključka celotnega preverjanja aplikacije.

Tabela 3 Rezultati lokalnih zbirk

Izvedena zbirka	Testi	Uspešni	Neuspešni
Osnovni projekt MJS	466	466	0
Next.js MJS	307	307	0
Next.js TypeScript	175	175	0
Skupaj	948	948	0

Vir: moj pregled in izvedeni preizkusi 16. 9. 2026



Grafikon 1 Rezultati izvedenih lokalnih testnih zbirk
Vir: moj preizkus oziroma zajem zaslona 16. 9. 2026

4.3 Uspešno testno plačilo

V razširjenem preizkusu sem ustvaril plačilni nalog za storitev s katalogizirano ceno 100 evrov. V zahtevi sem poslal vrednost amount nič, vendar sem v odgovoru preveril strežniško določen bruto znesek 10.000 centov. S tem sem že pri osnovnem toku preveril, da vhodnega zneska ne obravnavam kot avtoritativno ceno. Pozneje sem isto lastnost dodatno preveril z vrednostjo ena.

V sistemu Stripe sem preveril stanje succeeded. V rezervaciji sem preveril stanje plačila paid in povezavo s plačilno bremenitvijo. Pri zunanjem ponudniku sem preveril tudi povezavo s prenosom na povezani račun. V poslovnih podatkih sem našel dve finančni transakciji: razmerje med ponudnikom in potrošnikom ter provizijsko razmerje med platformo in ponudnikom. Poleg njiju sem preveril devet zapisov v finančni knjigi.

V sintetičnem primeru brez davka na storitev sem za osnovno transakcijo prebral neto in bruto znesek 10.000 centov ter davčni del nič. Pri proviziji sem prebral skupni znesek 1500 centov, od tega osnovo 1230 centov in davčni del 270 centov. Upravičenje ponudnika je znašalo 8500 centov. Te vrednosti sem uporabil za preverjanje notranje skladnosti konkretnega testnega primera, ne kot priporočilo za določanje provizij ali davkov.

Pri rezervaciji sem zabeležil tudi časovni in davčni kontekst. Test je preveril državo kupca, časovni pas uporabnika, različico davčnega pravila in razvrstitev storitve. Takšne podatke razumem kot posnetek okoliščin, v katerih sem sprejel plačilo. Če pozneje spremenim profil ponudnika, ne želim zaradi tega samodejno prepisati zgodovine že izvedene transakcije.

Ob pregledu rezultata sem videl stanje payment_finalization_status processing. Zato sem izvedel še naknadno branje in pregledal kodo prikaza stanja. V njej sem preveril, da lahko plačana rezervacija v tem stanju ostane do zaključka predvidenega finančnega postopka oziroma obdobja za vračilo. Samo iz te vrednosti zato nisem sklepal, da plačilo ni uspelo. Upošteval sem obstoj rezervacije, stanje paid in povezavo s sistemom Stripe.

4.4 Ponovna dostava dogodka

Za preverjanje idempotentnosti sem v sistemu Stripe poiskal dejanski dogodek payment_intent.succeeded, ki je pripadal izvedenemu testnemu plačilu. Pred ponavljanjem sem počakal, da je bil njegov prvi zapis označen kot obdelan. Nato sem shranil število povezanih rezervacij, transakcij, zapisov v finančni knjigi, računov in zapisov webhook dogodka. S tem sem pripravil stanje, s katerim sem primerjal rezultat ponovitev.

Isti dogodek sem nato trikrat dostavil zaporedno in desetkrat sočasno. Pri ponovnih dostavah sem uporabil veljaven podpis testne končne točke. Ne trdim, da je Stripe samodejno izvedel vseh trinajst ponovitev; ponovno sem poslal dejansko vsebino njegovega dogodka. To razlikovanje mi omogoča natančen opis preizkusa, hkrati pa še vedno preverim dejansko gostovano obdelavo.

V vseh ponovitvah sem dobil odgovor HTTP 200 z oznako podvojenega dogodka. Po preizkusu sem znova preveril poslovne količine. Ohranil sem eno rezervacijo, dve transakciji in devet zapisov v finančni knjigi. Ohranil sem tudi en obdelan zapis webhook dogodka. Predhodne in naknadne vrednosti so bile enake, zato sem scenarij ocenil kot uspešen.

Pri tej ugotovitvi ne ostanem pri splošnem zapisu, da je obdelava varna. Navedem natančno lastnost, ki sem jo preveril: trinajst dodatnih dostav istega dogodka ni ustvarilo dodatnih poslovnih učinkov v pregledanih zapisih. Ne sklepam o vseh mogočih prekinitvah procesa ali zaporedjih različnih tipov dogodkov. S takšno razmejitvijo ohranim zanesljiv, vendar stvaren rezultat izvedenega preizkusa.

4.5 Zavrnitev kartice

Za negativni plačilni scenarij sem uporabil pm_card_chargeDeclined. Najprej sem ustvaril nov plačilni nalog, nato pa v sistemu Stripe sprožil potrditev s testnim sredstvom za zavrnitev. Počakal sem, da je zaledje obdelalo rezultat, in preveril stanje naloga. Dobil sem stanje failed. Hkrati sem preveril, da nalog nima povezane rezervacije.

Za uspešen rezultat tega negativnega testa sem štel prav odsotnost rezervacije. Če bi ustvaril plačano rezervacijo kljub zavrnitvi, bi uporabniku in ponudniku posredoval napačno informacijo. Zato zavrnitev kartice v preglednici ne pomeni neuspeha testa. Test je uspel, ker sem pričakovano zavrnitev pravilno prenesel v poslovno stanje in nisem ustvaril učinka uspešnega plačila.

Pri razlagi ločim tudi sporočilo sistema Stripe in sporočilo aplikacije. Tehnični rezultat potrebujem za odločitev v zaledju, uporabniku pa moram ponuditi razumljivo nadaljevanje. V raziskavi sem ta scenarij zanesljivo izvedel prek API-ja. Prikaza konkretne zavrnjene kartice v brskalniku ne nadomestim s sliko drugega dogodka. Posnetke vmesnika zato označim glede na stanje, ki je na njih dejansko vidno.

4.6 Preklic in zahteva brez prijave

Pri dodatnem scenariju sem ustvaril nov testni plačilni nalog in ga pred uspešnim plačilom preklical v sistemu Stripe. Preveril sem stanje canceled, nato pa počakal na stanje cancelled v aplikacijski bazi. Nalogu ni bila pripisana rezervacija. S tem sem preveril dejansko povezavo med preklicnim dogodkom in notranjim stanjem, ne samo lokalne funkcije za spremembo besedila statusa.

Pri istem sklopu sem poslal zahtevo za ustvarjanje plačila brez uporabniškega žetona. Uporabil sem javni ključ projekta, vendar nisem dodal potrditve prijavljenega uporabnika. Prejel sem odgovor HTTP 401. Rezultat sem ocenil kot uspeh omejenega preverjanja avtentikacijske meje. Iz enega takšnega testa ne sklepam o vseh pravilih dostopa do vseh podatkovnih tabel.

Namerno sem preveril tudi manipulacijo s ceno. Vhodno vrednost amount sem nastavil na ena, odgovor pa je vseboval katalogizirano bruto ceno 10000 centov. S tem sem dobil konkreten dokaz, da strežnik pri tem scenariju ne uporablja predlaganega zneska kot cene storitve. Pri sklepanju o varnosti navedem to preverjeno lastnost, namesto da bi varnost pripisal samo uporabi strežniške tehnologije.

4.7 Podpisi in ločitev testnega načina

Na plačilni končni točki in končni točki Connect sem izvedel enak nabor mejnih preizkusov. Pri vsaki sem poslal pet pravilno podpisanih dogodkov z neobravnavanim tipom. Prejel sem pet odgovorov HTTP 200 z oznako, da je dogodek prezrt. S tem sem preveril sprejem veljavnega podpisa in obravnavo tipa, ki ne sme povzročiti poslovne spremembe.

Nato sem na vsako končno točko poslal dogodek brez podpisa, dogodek z napačnim podpisom in pravilno podpisan sintetični dogodek z oznako livemode true. Pri vseh šestih negativnih zahtevkih sem prejel odgovor HTTP 400. V celotnem sklopu sem tako izvedel šestnajst zahtevkov: deset pričakovano sprejetih in šest pričakovano zavrnjenih. Vsi so ustrezali določenim pričakovanjem.

Pri petih veljavnih dogodkih na plačilni končni točki sem izmeril mediano približno 283 milisekund in največjo izmerjeno vrednost približno 321 milisekund. Pri končni točki Connect sem izmeril mediano približno 315 milisekund in največjo vrednost približno 357 milisekund. Skripta pri tako majhnem vzorcu za zgornja percentila vrne največjo opazovano vrednost. Zato teh rezultatov ne predstavim kot stabilno napoved odzivnosti pri obremenitvi.

4.8 Vračilo, sprememba termina in Connect

Pri preizkusu vračila sem izvedel novo testno plačilo, nato pa kot uporabnik oddal zahtevo za vračilo. Kot sintetični ponudnik sem zahtevo odobril prek predvidene aplikacijske funkcije. V sistemu Stripe sem preveril dejansko ustvarjen testni objekt vračila in stanje succeeded. Znesek vračila je znašal 9800 centov, znesek pripadajoče pristojbine pa 200 centov.

V aplikacijskih podatkih sem preveril odobreno odločitev ponudnika, zaključen rezultat obdelave in nastanek zapisa za pristojbino vračila. Preveril sem tudi zapis dogodka refund_succeeded v vrsti za obvestila. Pri tem razlikujem med nastankom dogodka za obvestilo in dejansko dostavo sporočila v poštni predal. V diplomskem delu poročam o prvem, ker sem prav to preveril v podatkih.

Pri spremembi termina sem kot uporabnik predlagal nov čas za obstoječo rezervacijo. Preveril sem, da zabeležim predlagatelja in da spremembe ne označim kot že potrjene od ponudnika. V podatkih sem našel tudi dogodek spremembe termina. S tem sem preveril, da uporabniško dejanje ohrani sled izvora in ne odpravi potrebe po ustrezni nadaljnji potrditvi.

V razširjenem sklopu sem izvedel še testni spor in testno izplačilo Connect. Pri sporu sem preveril povezavo dogodka s sintetično rezervacijo ter stanje needs_response. Pri izplačilu sem preveril, da dogodek doseže ustrezno končno točko Connect in ohrani kontekst povezanega testnega računa. Izplačilo sem opazoval v stanju paid. Ta scenarija vključim kot razširitev preverjanja spremljajočih dogodkov plačilnega sistema.

Tabela 4 Rezultati integracijskih scenarijev

Scenarij	Kaj sem dejansko preveril	Izid
Uspešno plačilo	Rezervacijo, 2 transakciji in 9 finančnih zapisov	Uspešno
Ponovitve dogodka	3 zaporedne in 10 sočasnih dostav brez dodatnih učinkov	Uspešno
Sprememba termina	Predlog uporabnika in sled izvora	Uspešno
Vračilo	98,00 EUR vračila in 2,00 EUR pristojbine	Uspešno
Zavrnjena kartica	Stanje failed brez rezervacije	Uspešno
Testni spor	Povezavo z rezervacijo in needs_response	Uspešno
Izplačilo Connect	Povezani testni račun in stanje paid	Uspešno
Manipulacija cene	Vhod 1; odgovor 10.000 centov	Uspešno
Preklic	Stripe canceled in interni cancelled brez rezervacije	Uspešno
Brez prijave	Zavrnitev z odgovorom HTTP 401	Uspešno
Meje webhookov	10 sprejemov HTTP 200 in 6 zavrnitev HTTP 400	Uspešno

Vir: moj pregled in izvedeni preizkusi 16. 8. 2026

4.9 Dejanski uporabniški tok React

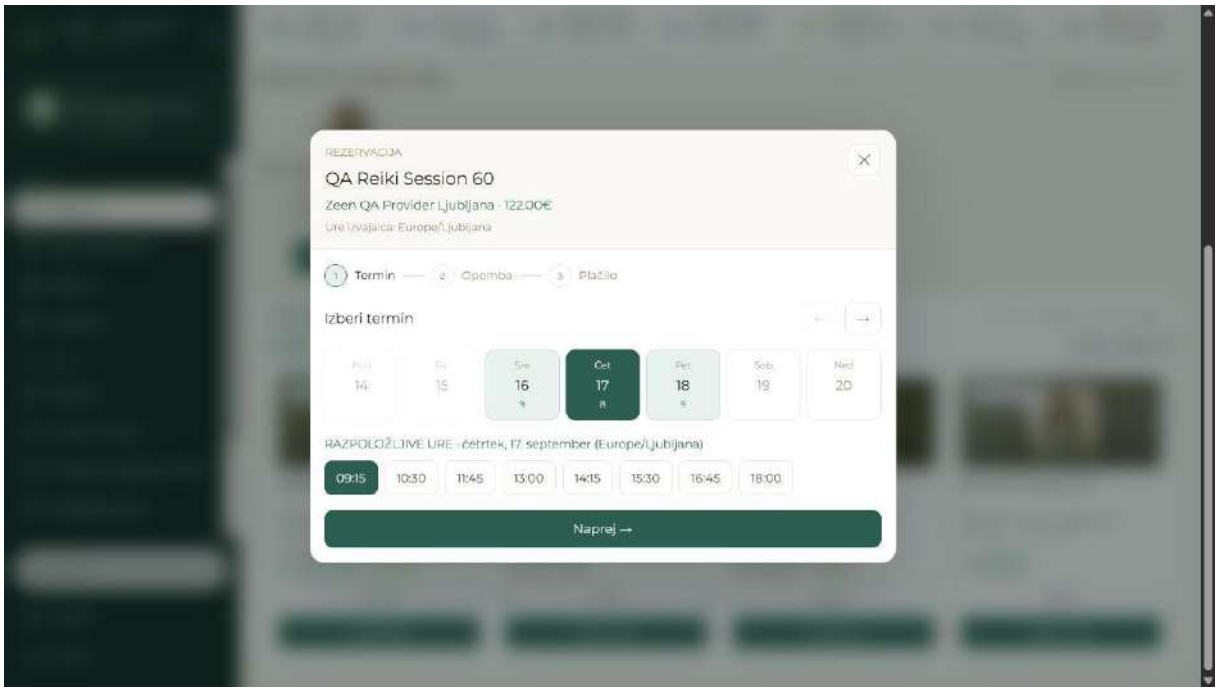
V zgrajeni aplikaciji React sem se prijavil s sintetičnim odjemalcem in na nadzorni plošči izbral storitev QA Reiki Session 60. Odprlo se je rezervacijsko okno s tremi fazami: termin, opomba

in plačilo. Na začetku sem preveril, da brez izbire termina ne morem nadaljevati. Po izbiri dneva sem dobil seznam razpoložljivih ur in izbral termin 17. septembra ob 9.15.

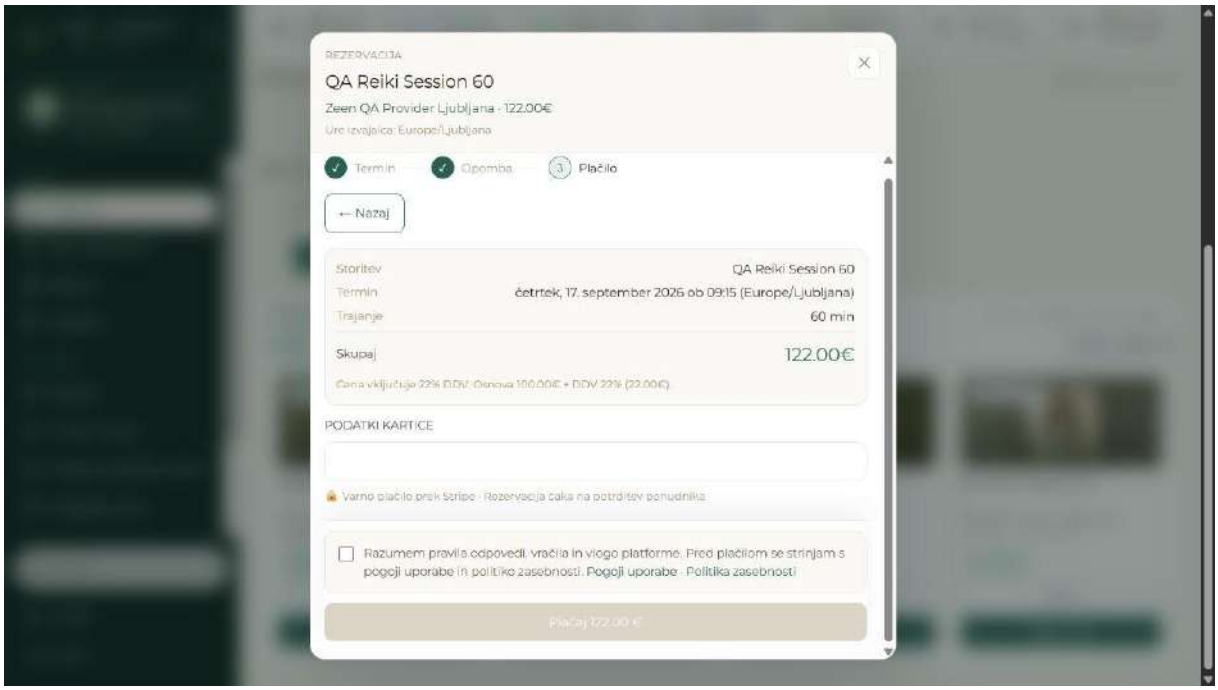
Pri prvem lokalnem naslovu na vratih 5175 sem dobil napako pri nalaganju razpoložljivosti. V konzoli sem preveril neuspešen neposredni klic zaledja, v konfiguraciji CORS pa sem našel dovoljene lokalne naslove na vratih 3000. Aplikacijo sem zato ponovno odprl na dovoljenem naslovu. Po tej uskladitvi sem termine uspešno naložil. Prvega neuspeha tako nisem napačno uporabil kot končni rezultat primerjave.

Ob napaki sem opazil tudi vsebinsko težavo: pod opozorilom o neuspešnem nalaganju sem dobil sporočilo, da ponudnik še nima urnika. Ker sem po pravilni nastavitvi termine dejansko pridobil, takšnega sporočila nisem ocenil kot ustrezno razlago omrežne napake. To opažanje sem vključil v hevristični pregled pomoči pri napakah. Pomembno se mi zdi tudi takrat, ko konfiguracijsko težavo odpravim.

V drugi fazi sem preveril prikaz izbranega termina in neobvezno polje za opombo. V tretji fazi sem pregledal povzetek storitve, trajanje in znesek. Vmesnik mi je prikazal 122 evrov ter pojasnilo, da cena vključuje 22 odstotkov DDV, pri čemer je osnovo prikazal kot 100 evrov. Nato sem uporabil gumb za vrnitev in preveril, da se vrnem na opombo z ohranjenim terminom.



Slika 1 Izbira termina v aplikaciji React
Vir: moj preizkus oziroma zajem zaslona 16. 8. 2026



Slika 2 Povzetek plačila v aplikaciji React s prikazom 122 evrov
Vir: moj preizkus oziroma zajem zaslona 16. 8. 2026

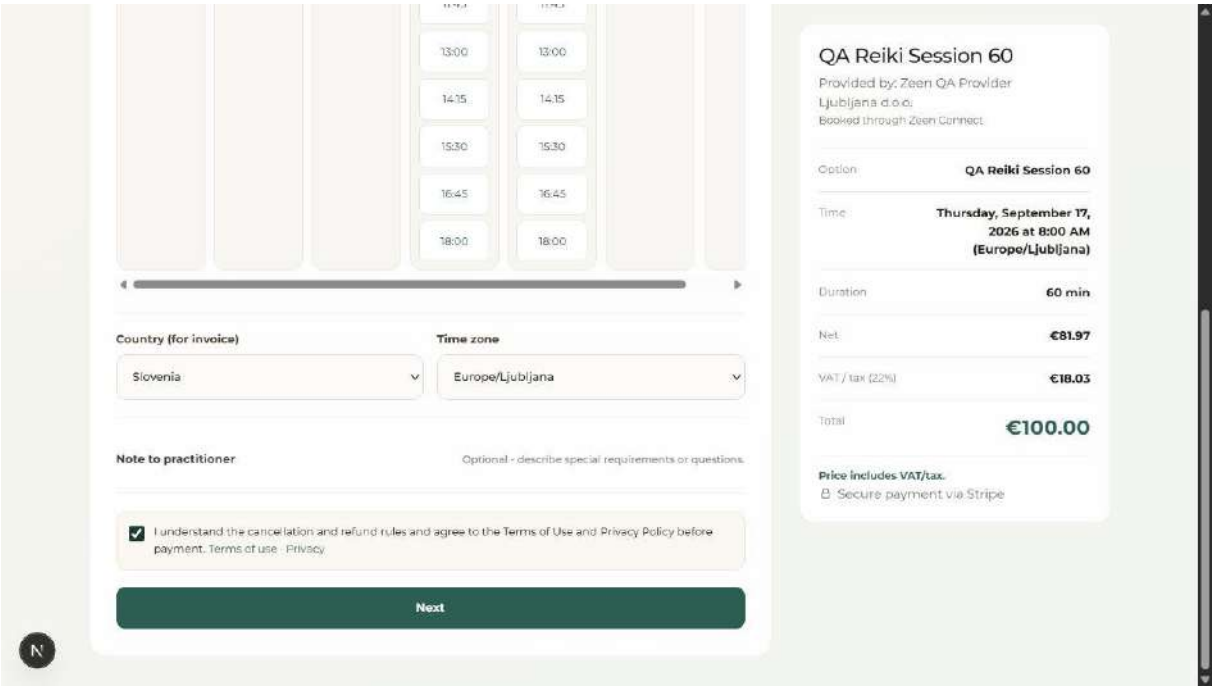
4.10 Dejanski uporabniški tok Next.js

V aplikaciji Next.js sem po prijavi izbral isto testno storitev. Namesto modalnega okna sem odprl samostojno rezervacijsko stran z dvema fazama: izbiro termina in podatki kartice. Pred izbiro sem preveril onemogočen gumb za nadaljevanje. Po izbiri termina sem videl osvežen povzetek z uro, časovnim pasom in trajanjem. Med pridobivanjem ponudbe sem dobil obvestilo o izračunavanju končne cene.

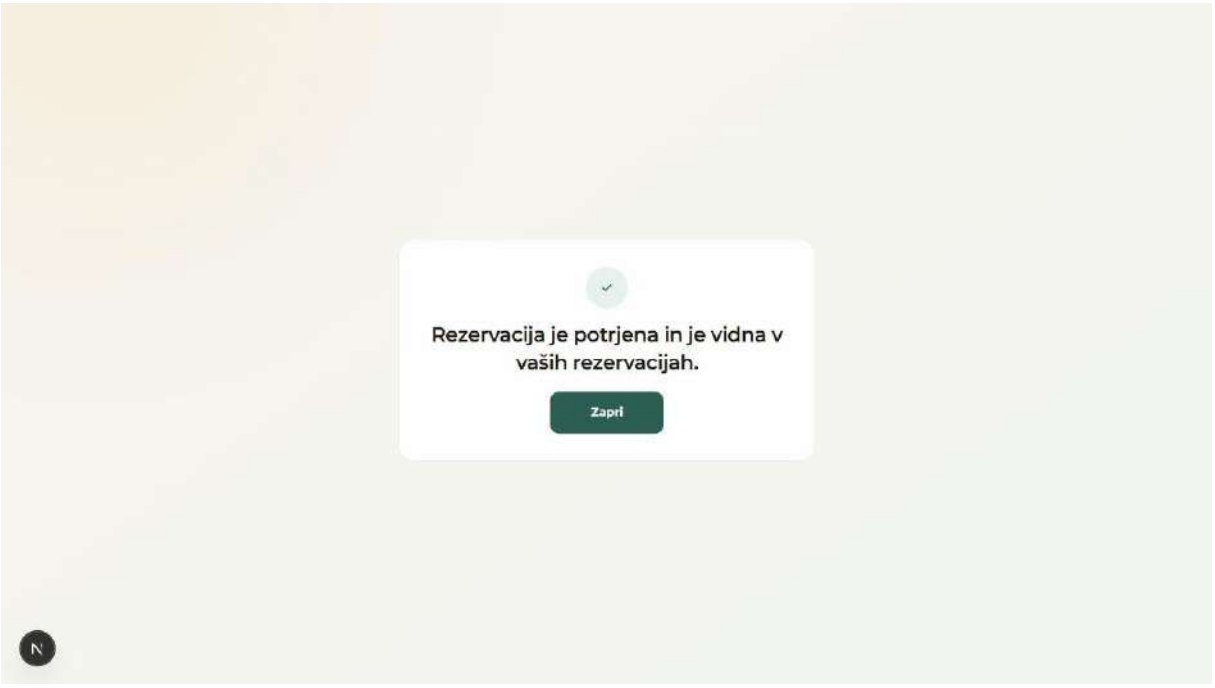
Ko je strežniški izračun prispel, sem v povzetku prebral neto znesek 81,97 evra, davčni del 18,03 evra in skupno ceno 100 evrov. Ta prikaz sem shranil kot dokaz cenovne razčlenitve. V primerjavi z Reactom sem opazil, da davek ni prištet katalogizirani ceni, temveč je razčlenjen znotraj nje. Ugotovitev sem povezal s testom avtoritativnega strežniškega zneska.

Po nadaljevanju sem odprl korak s kartičnim okvirjem Stripe. V uporabljenem brskalniškem okolju se polja v okvirju niso prikazala. Ob poskusu oddaje sem opazil onemogočen gumb z besedilom Loading, pri čemer se dejanje v opazovanem času ni zaključilo. To sem zabeležil kot negativen rezultat uporabniškega preizkusa nalaganja, ne kot neuspeh prej izvedenega plačila prek API-ja. Enako sem pri Reactu opazil prazen okvir, zato težave nisem pripisal samo eni izvedbi.

Za preverjanje potrditvenega zaslona sem nato odprl status osnovnega testnega plačila, pri katerem sem že preveril plačano rezervacijo. Stran je prebrala stanje iz zaledja in prikazala slovensko potrditev, da je rezervacija potrjena ter vidna med uporabnikovimi rezervacijami. Shranil sem dejanski zaslon. S tem sem ločeno preveril zaključni prikaz, ne da bi ustvaril izmišljeno potrdilo ali ročno spreminjal poslovni status.



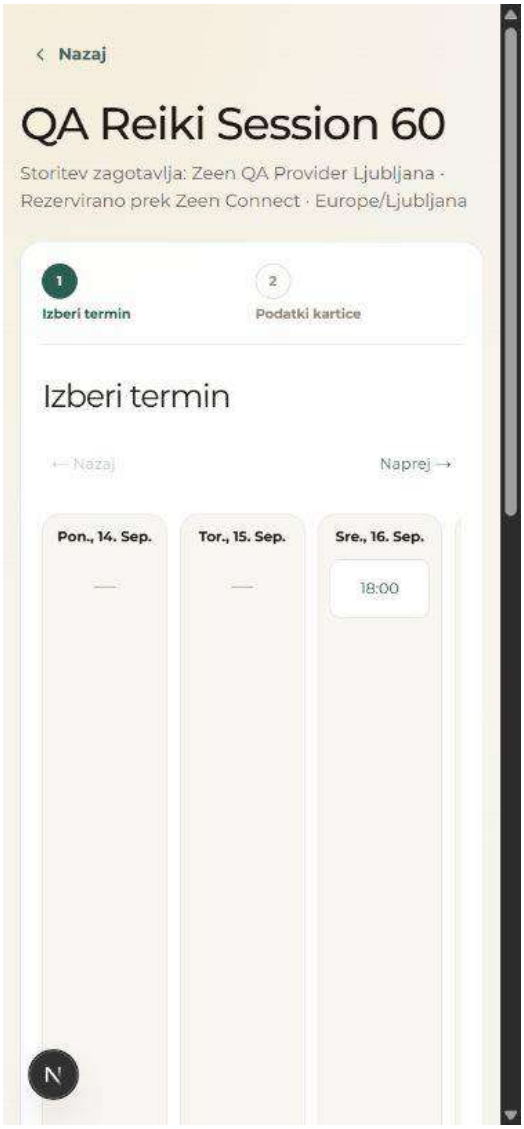
Slika 3 Strežniško izračunan povzetek v aplikaciji Next.js
Vir: moj preizkus oziroma zajem zaslona 16. 8. 2026



Slika 4 Prikaz potrditve dejansko plačane testne rezervacije
Vir: moj preizkus oziroma zajem zaslona 16. 8. 2026

4.11 Mobilni prikaz in semantični pregled

Pri obeh aplikacijah sem preveril tudi ožji prikaz s širino 390 slikovnih pik. Pri zajemu sem preveril širino dokumenta in nisem opazil, da bi bil celoten dokument širši od nastavljenega prikaznega območja. Ta rezultat uporabim kot omejen preizkus vodoravnega prelivanja. Ne enačim ga s preverjanjem vseh velikosti zaslona ali vseh naprav.



Slika 5 Dejanski mobilni prikaz izbire termina v Next.js
Vir: moj preizkus oziroma zajem zaslona 16. 8. 2026

Pri Reactu sem opazoval modalni prikaz nad nadzorno ploščo, pri Next.js pa samostojno stran. V drevesu elementov sem pri Reactu preveril tudi odsotnost izrecne vloge dialoga za rezervacijsko okno. Pri oznaki za zapiranje sem opazil simbol križca. Takšno opažanje uporabim kot izhodišče za izboljšanje semantike in poimenovanja; zgolj iz videza ne sklepam, da je okno primerno za vse načine uporabe.

Pri Next.js sem v pregledanem obrazcu našel oznake države, časovnega pasu in izbire termina ter ločeno označen povzetek. Na manjšem zaslonu sem moral vsebino pregledovati z navpičnim pomikom, kar pri obsežnejšem koledarju pričakujem. Pri oceni sem zato preverjal razumljivo zaporedje in ohranjene informacije, nisem pa zahteval, da so vse podrobnosti hkrati prikazane na enem zaslonu.

4.12 Zaključek izvedbe in ohranitev rezultatov

Po integracijskih preizkusih sem preveril, da skripte obnovijo začasno spremenjeni sintetični kontekst. V dokaznih datotekah sem ohranil rezultat obnove. Testne finančne dogodke sem pustil kot sled izvedbe v testnem okolju. Produkcijskih plačil nisem ustvarjal. Pri pripravi oddajnega paketa sem izločil dostopne podatke ter obdržal rezultate, slike in navodila za ponovitev z lastno konfiguracijo.

V praktičnem delu sem tako pridobil več različnih rezultatov. Potrdil sem delovanje izbranih zalednih scenarijev, dokumentiral razliko v uporabniški ceni in zabeležil težavo pri nalaganju kartičnega okvirja v uporabljenem brskalniku. Nobenega od teh rezultatov nisem prilagodil pričakovanemu zaključku. Prav njihova kombinacija mi omogoča, da v nadaljevanju ločim tehnično zanesljivost skupnega zaledja od kakovosti konkretnega uporabniškega toka.

5 PRIMERJALNA ANALIZA IN VREDNOTENJE

Pri primerjavi združim pregled kode, rezultate testov in opažanja vmesnika. Najprej ločim odgovornosti, ki so skupne obema izvedbama, od tistih, ki jih dejansko primerjam. S tem preprečim, da bi uspeh ene zaledne funkcije prikazal kot prednost dveh različnih uporabniških tehnologij. Nato ocenim konkretne razlike in njihov pomen za uporabnika, razvijalca in vzdrževalca.

5.1 Razporeditev odgovornosti

V Reactu najdem neposredno povezavo med stanjem rezervacijskega okna in zahtevki v Supabase. V isti uporabniški poti spremljam izbiro termina, začasno stanje obrazca, klic razpoložljivosti in začetek plačila. Prednost takšne organizacije vidim v neposrednosti: pri posameznem koraku lahko hitro najdem dejanje in spremembo prikaza. Pri večjem obsegu pa moram paziti, da v komponento ne prenesem preveč poslovnih pravil.

V Next.js med uporabniško komponento in Supabase vključim strežniško akcijo. V datoteki booking-actions.ts lahko pregledam preverjanje vhodnih podatkov, ustvarjanje ponudbe in zahtevo za plačilni nalog. V komponenti se nato osredotočim na uporabniško stanje ter prikaz rezultata. Takšna razdelitev mi olajša sledenje meji med zaupanja vrednim izračunom in podatki, ki jih uporabnik pošlje iz brskalnika.

Iz te razlike ne sklepam, da pri Reactu ne morem doseči podobne razdelitve. Dosegel bi jo lahko z dodatnimi moduli in dosledno uporabo zalednih odgovorov. Prav tako lahko v Next.js ustvarim nepregledno komponento, če odgovornosti pomešam. Pri ugotovitvi zato ostanem pri pregledani kodi: v obravnavanem projektu mi struktura Next.js bolj neposredno pokaže strežniško plast rezervacijskega toka.

Skupno zaledje mi omogoča pomemben nadzor pri primerjavi. Pravila za ustvarjanje plačila in obdelavo dogodka ostanejo enaka, če zamenjam uporabniško aplikacijo. Tako razliko v uporabniški ceni lažje pripišem prikazu oziroma pripravi podatkov v aplikaciji. Če bi primerjal dve popolnoma različni zaledji, bi moral pred takšnim sklepom preveriti še razlike med katalogoma, davčnima nastavitvama in obdelavo rezervacij.

5.2 Cena in pogodba med plastmi

Najizrazitejšo vsebinsko razliko najdem pri ceni. V Reactu preberem 122 evrov, v Next.js pa 100 evrov. Strežniški test pri isti katalogizirani ceni vrne 10.000 centov. Razlika zato presega oblikovanje valute ali zaokroževanje. V Reactu znesek obravnavam kot osnovo, ki ji prištejem davek, v Next.js pa prikažem razčlenitev znotraj končne cene.

Pri uporabniku bi takšno neskladje zmanjšalo zaupanje tudi takrat, ko zaledje dejansko bremeni pravilen znesek. Če na gumbu prebere 122 evrov, pozneje pa vidi plačilo 100 evrov, težko razume, ali je dobil popust, delno plačilo ali pa je prišlo do napake. Zato pravilne bremenitve ne štejem za dovolj dobro rešitev napačnega prikaza. Pred potrditvijo mora uporabnik videti isti končni znesek, ki ga namerava plačati.

Kot priporočeno spremembo določim uporabo enega strežniškega odgovora za celoten cenovni povzetek. Iz njega bi prebral osnovo, davčni del, valuto in skupni znesek. V brskalniku bi nato oblikoval le prikaz. S tem bi zmanjšal možnost, da dve aplikaciji različno razumeta isto katalogizirano polje. Predvsem bi se izognil vzporednemu vzdrževanju davčne aritmetike v uporabniški komponenti in zaledju.

Pri tej ugotovitvi upoštevam čas preizkusa. Davčni profil sem pred primerjavo zaslonov obnovil, zato ne primerjam zaslona s stopnjo nič in zaslona s stopnjo 22 odstotkov. V besedilu ohranim ločitev med sintetičnim integracijskim primerom in poznejšim pregledom vmesnika. S tem preprečim napačno razlago, da se je razlika pojavila samo zato, ker sem med primerjavo spreminjal vhodne pogoje.

5.3 Konfiguracija in mesto omrežne povezave

Pri pripravi Reacta sem moral uskladiti izvor brskalnika s pravili zaledja. Začetni lokalni naslov na drugih vratih ni omogočil uspešnega klica razpoložljivosti. Ko sem uporabil dovoljeni naslov na vratih 3000, sem termine pridobil. V Next.js sem isti podatek pridobil prek strežniške akcije. To opažanje razumem kot posledico različnega mesta povezave, ne kot dokaz, da bi moral pravila CORS odstraniti.

Z vidika razvijalca mi neposredni brskalniški klic omogoča pregled zahteve v uporabniškem delu, hkrati pa moram pravilno nastaviti dovoljene izvore. Pri strežniški akciji se del odgovornosti preseli v aplikacijski strežnik. Tam moram pravilno upravljati uporabniško sejo, okoljske spremenljivke in odgovor zunanje funkcije. Konfiguracija torej ne izgine; spremeni se njena razporeditev.

Zato ne navedem izmišljene skupne razlike v številu konfiguracijskih korakov. Pri preizkusu sem uporabljal že vzpostavljeno okolje, ne dveh novih namestitev od začetka. Lahko opišem dejansko opravljeno prilagoditev lokalnega naslova in pregledane povezave, ne morem pa teh dejanj ustrezno predstaviti kot celoten strošek začetne uvedbe. Za vprašanje zahtevnosti uporabim pregled odgovornosti in sled izvedenih dejanj.

Ta razmejitev mi pomaga tudi pri ponavljanju raziskave. Če drug razvijalec uporabi druga vrata, lahko dobi drugačen začetni rezultat, čeprav uporablja isto kodo. Zato v navodilu poleg ukaza za zagon navedem tudi naslov, na katerem sem vmesnik preveril. Ponovljivost razumem kot ujemanje kode, podatkov in pomembnih nastavitev, ne samo kot ponovno izvajanje istega ukaza.

5.4 Razumevanje uspeha in negativnega testa

Pri rezultatih jasno ločim med uspehom plačila in uspehom testa. Test zavrnitve kartice uspe, ko plačilo ne uspe na pričakovani način in ne ustvari rezervacije. Test manjkajočega podpisa uspe, ko zahtevo zavrnem. Če bi vse odgovore HTTP 400 ali HTTP 401 označil kot neuspeh, bi napačno ovrednotil prav zaščitno vedenje, ki sem ga želel preveriti.

Obratno pa HTTP 200 še ne pomeni pravilnega poslovnega učinka. Pri ponovitvi dogodka pričakujem uspešen odgovor brez nove rezervacije. Pri neobravnavanem dogodku pričakujem varen izhod brez spremembe. Zato k vsakemu tehničnemu odgovoru pripišem pričakovani pomen. Ta pristop mi omogoča, da tabelo rezultatov preberem brez ugibanja, ali beseda uspešno pomeni transakcijo, zavrnitev ali samo zaključen zahtevek.

Podobno ravnam pri vmesniku. Prazen kartični okvir in obstal gumb sem zabeležil kot izveden negativen uporabniški preizkus. Prej potrjeno plačilo prek API-ja zaradi tega ni postalo neuspešno, celoten uporabniški tok pa tudi ni postal dokazan samo zato, ker je API deloval. S tem ohranim oba rezultata in usmerim nadaljnjo diagnostiko na mesto, kjer se dejansko razlikujeta.

5.5 Hevristični pregled

Hevristični pregled sem izvedel na odprtih zaslonih prijave, izbire termina, povzetka, plačilnega koraka in potrditve. Pri Reactu sem dodatno opazoval sporočilo o omrežni napaki med pripravo lokalnega okolja. V preglednici za vsako načelo navedem svoje dejansko opažanje. Kjer sem preveril le prisotnost povezave ali oznake, tega ne razširim v trditev o izvedbi celotne naloge.

Pri vidnosti stanja sem v obeh aplikacijah prepoznal korake, pri Next.js pa še stanje izračunavanja ponudbe in potrditveno stran, vezano na podatke zaledja. Slabost sem opazil ob praznem kartičnem okvirju: uporabnik ne dobi dovolj jasne razlage, zakaj ne more nadaljevati. Zato isto načelo ocenim z navedbo uspešnega in problematičnega dela, namesto da bi celotni aplikaciji pripisal eno nedoločeno oznako.

Pri nadzoru uporabnika sem se v Reactu dejansko vrnil iz plačila na opombo in preveril ohranitev termina. V Next.js sem na plačilnem koraku pregledal možnost vrnitve, na potrditveni strani pa povezavo do rezervacij. Pri učinkovitosti sem primerjal vidne faze: tri v Reactu in dve v Next.js. S tem ne trdim, da je vsak uporabnik v Next.js hitrejši, saj časa dela več uporabnikov nisem meril.

Pri doslednosti sem največjo težo namenil ceni. V istem poslovnem primeru mora biti znesek pojasnjen enako v kartici storitve, povzetku in plačilu. Pri prepoznavanju informacij sem preveril prisotnost storitve, termina in trajanja. Na potrditvenem zaslonu Next.js sem sicer dobil jasno potrditev, vendar brez podrobnega povzetka storitve. Kot izboljšavo zato predlagam, da zaključek povežem še z osnovnimi podatki rezervacije.

Pri pomoči sem preveril povezave do pogojev in zasebnosti ter besedilo ob napaki. Povezav do pravnih pojasnil ne enačim z neposredno podporo pri tehnični težavi. Če se kartični element ne naloži, bi uporabniku ponudil konkreten ponovni poskus in ustrezno pot do pomoči. Pri sporočilu o urniku pa bi ločil potrjeno odsotnost terminov od neuspešnega prenosa podatkov.

Tabela 5 Rezultati dokumentiranega hevrističnega pregleda

Hevristika	**Moje opažanje v React**	**Moje opažanje v Next.js**
Vidnost stanja	Vidim 3 faze in onemogočeno nadaljevanje brez termina.	Vidim 2 fazi, izračun cene in potrditev rezervacije.
Ujemanje z realnim svetom	Vidim storitev, uro in 122 EUR; znesek se ne ujema z zaledjem.	Vidim storitev, uro in razčlenitev skupnih 100 EUR.
Nadzor in svoboda	Z vrnitvijo na opombo ohranim izbrani termin.	Na plačilu vidim vrnitev, na potrditvi povezavo do rezervacij.
Doslednost	Ugotovim različno razumevanje katalogizirane cene.	Prikazano končno ceno povežem s strežniškim izračunom.
Preprečevanje napak	Brez termina ne nadaljujem; plačilo zahteva dodatno potrditev.	Brez termina ne nadaljujem; med izračunom počakam.
Prepoznavanje	V povzetku preberem storitev, termin in trajanje.	Povzetek ostane viden; na končni potrditvi pogrešam podrobnosti.
Učinkovitost	Opravim ločen korak z neobvezno opombo.	Opombo najdem v prvem koraku; vidim manj faz.

Hevristika	Moje opažanje v React	Moje opažanje v Next.js
Preglednost	Nalogo spremljam v osrednjem modalnem oknu.	Nalogo spremljam na samostojni strani s povzetkom.
Pomoč pri napakah	Ob omrežni napaki dobim tudi zavajajočo navedbo o urniku.	Ob praznem kartičnem okvirju opazim nalaganje brez jasne rešitve.
Pomoč in pojasnila	Najdem pogoje in zasebnost; pri nalaganju potrebujem jasnejšo pomoč.	Najdem pogoje in zasebnost; potrebujem jasnejši ponovni poskus.

Vir: moj pregled in izvedeni preizkusi 16. 8. 2026

5.6 Prednostne izboljšave

Na prvo mesto postavim uskladitev cenovnega prikaza Reacta s strežniškim izračunom. Spremembo bi preveril s primerom bruto cene 100 evrov, pri katerem mora biti vsota osnovnega in davčnega dela enaka prikazanemu skupnemu znesku. Nato bi preveril še primer brez davčnega dela. Za uspeh spremembe bi zahteval enak znesek v povzetku, na gumbu in v odgovoru ustvarjenega plačilnega naloga.

Na drugo mesto postavim obravnavo nepripravljenega kartičnega elementa. Pred omogočanjem oddaje bi preveril, ali je element pripravljen. Ob predolgem nalaganju bi prikazal razumljivo sporočilo in možnost ponovnega poskusa. Pri tem ne bi trdil, da je kartica zavrnjena, saj v opaženem primeru kartičnih podatkov še nisem oddal. Razliko med nalaganjem, nepopolnim vnosom in zavrnitvijo bi ohranil tudi v zapisniku napak.

Na tretje mesto postavim semantiko modalnega okna Reacta in natančnost sporočil. Okno bi opremil z ustreznim poimenovanjem in načinom upravljanja fokusa. Gumb s križcem bi razumljivo poimenoval. Pri napaki pri pridobivanju razpoložljivosti bi uporabniku povedal, da podatkov trenutno ne morem prikazati, ne pa, da ponudnik zagotovo nima urnika. Takšne izboljšave neposredno povežem z opazovanimi zasloni.

Predloge jasno ločim od izvedenih sprememb. V okviru te raziskave sem preverjal in dokumentiral obstoječi aplikaciji ter pripravil dopolnjen raziskovalni paket. Zato priporočenih programskih sprememb ne označim kot že uvedene popravke. Njihovo vrednost utemeljim s konkretnim sprožilcem, pričakovanim vedenjem in načinom, kako bi preveril uspešnost popravka.

5.7 Vrednotenje hipotez

Hipotezo H1 v opredeljenem obsegu potrdim. V Reactu sem opazoval tri uporabniške faze, v Next.js pa dve. V pregledani kodi Next.js sem zaledne klice spremljal prek strežniških akcij,

medtem ko sem v Reactu spremljal neposredne klice iz brskalnika. Hipoteze ne razširim v trditev o točno določenem prihranku razvojnih ur ali manjšem številu vseh konfiguracijskih nastavitev.

Hipotezo H2 potrdim za izvedene scenarije skupnega zaledja. Preveril sem uspešno plačilo, zavrnitev, preklic, strežniško določitev cene, zavrnitev zahteve brez prijave, obravnavo podpisov in odsotnost dodatnih učinkov ob ponovitvah. Rezultat dopolnim z uspešnim vračilom, spremembo termina ter testnima dogodkoma spora in izplačila. Pri potrditvi poudarim obseg, saj s tem ne podajam splošnega zagotovila o odsotnosti vseh ranljivosti.

Hipotezo H3 podprem delno. Pri Next.js sem opazil manj vidnih faz in pravilen prikaz končne cene za izbrani primer. Pri potrditvi sem dobil razumljivo sporočilo, povezano z dejansko rezervacijo. Hkrati sem opazil prazen kartični okvir in premalo jasno nadaljevanje ob njegovem nalaganju. Zato ne trdim, da sem dokazal brezhibno uporabniško izkušnjo ali splošno prednost pri vseh nalogah.

Tabela 6 Vrednotenje hipotez

Hipoteza	**Osnova za mojo presojo**	**Presoja**
H1	Primerjam 3 faze Reacta z 2 fazama Next.js in mesto zalednih klicev.	Potrdim v opredeljenem obsegu.
H2	Preverim ceno, podpise, zavrnitve, preklic in nespremenjene učinke ponovitev.	Potrdim za izvedene scenarije.
H3	V Next.js dobim preglednejšo ceno in manj faz; zabeležim tudi težavo kartičnega okvirja.	Podprem delno.

Vir: moj pregled in izvedeni preizkusi 16. 8. 2026

5.8 Pomen skupnega zaledja

Skupno zaledje mi v raziskavi prinaša dve prednosti. Prva je primerljivost poslovnih podatkov. Druga je možnost, da napako v prikazu ločim od napake v plačilnem pravilu. Če pošljem znesek ena, strežnik pa vrne katalogizirano ceno, imam jasen dokaz o meji zaupanja. Če nato vmesnik prikaže drug znesek, lahko nadaljnji pregled usmerim v način oblikovanja uporabniškega povzetka.

Hkrati moram priznati tudi omejitev takšne zasnove. Pri uspešnem preizkusu webhooka ne primerjam dveh neodvisnih implementacij podpisa. Preverjam eno skupno implementacijo, ki služi obema aplikacijama. Rezultat zato uporabim kot pogoj pravilnega delovanja plačilnega sistema in kot skupno osnovo, ne kot tekmovanje med Reactom in Next.js pri kriptografskem preverjanju.

Ta ugotovitev mi spremeni tudi način razmišljanja o migraciji. Če uporabniški del prenesem v Next.js, ni nujno, da hkrati prepišem celotno plačilno logiko. Lahko ohranim preverjene zaledne funkcije in najprej uskladim uporabniške pogodbe, cenovni prikaz ter obravnavo napak. Tak postopen pristop presojam kot možnost, ki jo opažam v pregledani arhitekturi, ne kot edino pravilno pot za vsak projekt.

5.9 Vzdrževanje in odpravljanje napak

Pri vzdrževanju mi najbolj pomaga možnost, da posamezni problem povežem z naročilom in dogodkom. Če uporabnik sporoči, da vidi nalaganje, želim vedeti, ali je plačilni nalog že nastal, ali ga je Stripe potrdil in ali je zaledje ustvarilo rezervacijo. Brez teh povezav bi lahko uporabniku predlagal ponovni poskus v trenutku, ko je plačilo že uspešno, in s tem povečal zmedo.

V pregledanem primeru sem lahko osnovni test povezal s statusno stranjo. To povezavo ocenim kot pomembnejšo od same prisotnosti zelene ikone. Potrditev je bila uporabna zato, ker je izhajala iz preverjenega stanja. Za nadaljnje vzdrževanje bi enako sledljivost uporabil tudi pri negativnih dogodkih: zabeležil bi vrsto napake, fazo postopka in identifikator naloga, pri tem pa iz dnevnikov izključil skrivne vrednosti.

Pri razlagi stanja processing sem moral pregledati poslovni pomen, ne samo imena polja. To izkušnjo razumem kot opozorilo pri načrtovanju podatkovnih modelov. Če podobno besedo uporabim za čakanje na plačilo in za poznejši finančni postopek, lahko vzdrževalec ali uporabnik napačno razume stanje. Zato bi poleg notranjih oznak ohranil jasna pravila, kako iz kombinacije podatkov sestavim uporabniško sporočilo.

5.10 Veljavnost ugotovitev

Notranjo veljavnost raziskave podpiram z uporabo iste testne storitve, skupnega zaledja in dokumentiranih vhodnih pogojev. Kjer sem pogoje začasno spremenil, sem to navedel in preveril njihovo obnovitev. Prav tako sem začetno napako lokalnega izvora ponovno preveril na dovoljenem naslovu. Tako sem zmanjšal možnost, da bi končni sklep temeljil na napačni konfiguraciji namesto na lastnosti pregledane izvedbe.

Zunanjo veljavnost omejim na obravnavani projekt in datum izvedbe. Druga različica iste kode lahko vsebuje drugačne popravke, drug brskalnik pa lahko drugače naloži zunanji element. Zato ohranim dokazne datoteke in posnetke, ki predstavljajo dejansko opazovanje. Ne pričakujem,

da bi rezultat posameznega zaslona brez ponovnega preverjanja veljal za vse prihodnje različice aplikacije.

Pri strokovni presoji uporabnosti upoštevam lastno vlogo ocenjevalca. Ker poznam tehnično ozadje, lahko lažje razumem izraze, ki bi uporabnika zmedli. Zato v preglednici namesto zgolj številčne ocene zapišem opažanje in njegovo posledico. Tak zapis bralcu omogoča presojo, ali se z mojo razlago strinja, in predstavlja osnovo za poznejše testiranje z dejanskimi uporabniki.

5.11 Kaj bi spremenil v naslednjem preverjanju

V naslednjem preverjanju bi najprej ponovil celoten uporabniški tok po uskladitvi cenovnega prikaza in obravnave pripravljenosti kartičnega elementa. Določil bi enako nalogo za oba vmesnika: izbiro storitve, prostega termina, pregled cene, oddajo testnega plačila in preverjanje rezervacije. Pri vsakem koraku bi shranil čas, rezultat in prehod v naslednje stanje. Tako bi dopolnil sedanje ločeno preverjanje API-ja in uporabniških zaslonov.

Nato bi vključil več uporabnikov z različnimi izkušnjami. Pri njih bi meril dokončanje naloge, napake, pomoč, ki jo potrebujejo, ter razumevanje cene in potrditve. Takšna meritev bi odgovorila na vprašanja, ki jih s svojim hevrističnim pregledom samo odprem. Posebej bi preveril, ali uporabnik razume razliko med plačano rezervacijo in terminom, ki še čaka na izvajalčevo odločitev.

Za tehnično razširitev bi pripravil še nadzorovane prekinitve med nastankom plačila in zapisom rezervacije ter ponovitve po ponovnem zagonu obdelave. Pri tem bi ohranil ista merila glede števila poslovnih učinkov. Sedanjega uspeha sočasnih ponovitev ne bi uporabil kot nadomestilo za preizkus vsake prekinitvene točke. Nadaljnji korak bi zato usmeril v novo tveganje, ki ga dosedanji scenariji ne pokrivajo, namesto da bi samo večkrat ponavljal že uspešno preverjanje.

5.12 Odgovor na primerjalno vprašanje

Na osnovi zbranih dokazov ugotavljam, da pri obravnavani izvedbi Next.js pridobim preglednejšo povezavo uporabniške akcije s strežniškim izračunom in krajšo vidno razdelitev rezervacije. Pri Reactu ohranim neposreden tok v modalnem oknu, vendar sem našel neskladje cenovnega prikaza in dodatno občutljivost na nastavitev izvora. Te razlike pripišem pregledani implementaciji in organizaciji odgovornosti.

Največjo skupno vrednost prepoznam v zalednih pravilih, ki so prestala dejanske testne transakcije in negativne preizkuse. Pravilno strežniško določanje cene, preverjanje dogodkov

in preprečevanje podvajanja mi omogočajo zanesljivo osnovo za obe uporabniški aplikaciji. Kakovost končnega izdelka pa dosežem šele, ko tej osnovi dodam pravilen prikaz in jasno ravnanje ob težavi. To povezavo štejem za glavno primerjalno ugotovitev svojega dela.

5.13 Razlaga finančnih zapisov

Pri pregledu finančnih dokazov sem moral razlikovati med številom poslovnih transakcij in številom posameznih knjižnih zapisov. Dve transakciji ter devet zapisov ne pomenijo, da sem uporabnika devetkrat bremenil. V testnem primeru sem preverjal različna poslovna razmerja in njihove spremljajoče učinke. Zato sem število zapisov uporabil predvsem kot primerjalno merilo pred ponovitvijo dogodka in po njej, ne kot neposredno število plačil uporabnika.

Enako previdno sem razložil vrednost nič pri številu končnih računov v zgodnjem posnetku. Pri uspešnem testu sem preveril plačilo, rezervacijo in druge finančne zapise, koda pa omogoča poznejši zaključek računskega postopka. Zato iz ničelne vrednosti nisem izpeljal trditve, da račun v aplikaciji nikoli ne nastane. V preglednici sem ohranil dejansko opazovano količino, njeno razlago pa povezal s fazo življenjskega cikla.

Pri vračilu sem primerjal znesek objekta Stripe z vrednostjo, ki jo je določila aplikacija. S tem sem preveril skladnost dveh sistemov. Zgolj zapis odobreno v uporabniškem delu ne bi dokazal, da je zunanja operacija uspela. Prav tako bi mi samo ustvarjeni objekt vračila povedal premalo o tem, ali je aplikacija odobritev pravilno povezala z rezervacijo. Za sklep sem potreboval obe strani povezave.

Ta način branja dokazov bi uporabil tudi pri obravnavi uporabnikove prijave težave. Najprej bi ugotovil, katero dejanje je uporabnik izvedel in katero stanje je pričakoval. Nato bi pregledal povezane zapise in šele zatem predlagal popravek. Pri plačilnem postopku bi se izognil popravljanju zgolj vidnega besedila, če bi težava izvirala iz napačne povezave podatkov. S tem bi ohranil vsebinsko pravilnost sistema in zmanjšal možnost, da bi navidezno rešena težava povzročila novo neskladje pri vračilu ali poznejšem zaključku.

6 SKLEP

V diplomskem delu sem primerjal integracijo sistema Stripe v aplikacijo React s Supabase in v aplikacijo Next.js. Za primerjavo sem uporabil dejanski projekt Zeen Connect, skupno testno zaledje in sintetične uporabniške račune. Raziskavo sem razširil od pregleda kode do izvedenih testnih transakcij ter dejanskih uporabniških zaslonov. Tako sem lahko preveril razlike, ki jih samo iz arhitekturnega opisa ne bi mogel zanesljivo ugotoviti.

Na prvo raziskovalno vprašanje odgovarjam, da mi v pregledani izvedbi Next.js strežniške akcije olajšajo sledenje povezavi med uporabniškim dejanjem in zaledno operacijo. Pri Reactu sem spremljal neposredne klice iz brskalnika ter tri vidne faze rezervacije. Pri Next.js sem spremljal dve fazi in strežniško pridobljen cenovni povzetek. Takšna razlika mi pokaže možnost bolj strnjenega uporabniškega postopka, vendar je ne enačim s samodejno manjšo zahtevnostjo celotnega sistema. Na drugo raziskovalno vprašanje odgovarjam z rezultati dejanskih preizkusov. Uspešno sem izvedel osnovno in razširjeno testno plačilo, preveril zavrnitev kartice, preklic, avtoritativno strežniško ceno ter zavrnitev zahteve brez prijave. Preveril sem tudi podpise na dveh končnih točkah webhook. Pri ponovitvi dejanskega dogodka sem po treh zaporednih in desetih sočasnih dostavah ohranil enako število poslovnih učinkov. S tem sem za izbrani primer podprl pravilnost povezave med plačilom in rezervacijo.

Pri vračilu sem preveril konkreten rezultat v sistemu Stripe in aplikaciji. Od testnega plačila v višini 100 evrov sem v izbranem postopku dobil vračilo v višini 98 evrov ter zapis pristojbine v višini 2 evrov. Preveril sem tudi spremembo termina s sledjo predlagatelja, povezavo testnega spora in dogodek testnega izplačila Connect. Ti rezultati so mi omogočili širši pogled na plačilni življenjski cikel, ki se ne konča s prvim uspešnim plačilom.

Na tretje raziskovalno vprašanje odgovarjam z dokumentiranim hevrističnim pregledom. Pri obeh aplikacijah sem ugotovil uporaben prikaz korakov in povzetka storitve. V Reactu sem preveril ohranitev termina pri vrnitvi iz plačilnega koraka. V Next.js sem preveril strežniško razčlenjeno ceno in prikaz potrditve dejansko plačane rezervacije. Ob tem sem prepoznal tudi pomanjkljivosti, ki jih ne bi mogel razbrati iz samega števila uspešnih avtomatiziranih testov.

Najpomembnejša vsebinska pomanjkljivost je bila razlika v prikazani ceni. Pri isti storitvi sem v Reactu videl 122 evrov, v Next.js in odgovoru zaledja pa 100 evrov. Iz tega sem izpeljal priporočilo, da cenovni povzetek v obeh aplikacijah oprem na en strežniški izračun. Po moji presoji je pravilno pojasnjena končna cena temelj zaupanja pri plačilu; zanesljivo zaledje ne

odpravi posledic napačnega zneska na uporabniškem gumbu. Pomembno ugotovitev sem dobil tudi pri pripravi okolja. Napako pri nalaganju terminov ob prvem lokalnem zagonu Reacta sem odpravil z uporabo dovoljenega izvora na vratih 3000. S tem sem se izognil napačnemu sklepu, da aplikacija ne podpira izbire termina. Hkrati sem ohranil opažanje o neustreznem sporočilu, ki je omrežno težavo povezalo z odsotnostjo urnika. V nadaljnjem razvoju bi te vzroke uporabniku pojasnil ločeno.

Pri kartičnem koraku sem v uporabljenem brskalniškem okolju opazil prazen zunanji okvir. Pri poskusu nadaljevanja v Next.js sem dobil stanje nalaganja brez jasnega zaključka. Ta rezultat sem vključil v raziskavo, čeprav so dejanske testne transakcije prek API-ja uspele. Tako sem ohranil razliko med pravilnim plačilnim zaledjem in dokončno preverjenim uporabniškim vnosom kartice. Za naslednji razvojni korak zato določim izboljšanje zaznave pripravljenosti elementa in razumljivega ravnanja ob napaki pri nalaganju. Hipotezo H1 sem v opredeljenem obsegu potrdil s primerjavo faz in mesta zalednih klicev. Hipotezo H2 sem potrdil za izvedene scenarije skupnega zaledja. Hipotezo H3 sem podprl delno: pri Next.js sem dobil preglednejšo ceno in manj faz, vendar zaradi opaženih težav celotnega uporabniškega toka nisem označil za brezhibnega. Takšno vrednotenje razumem kot rezultat raziskave, pri kateri hipotezo presodim po dokazih in ne po prvotnem pričakovanju.

Kot praktičen rezultat dela sem pripravil ponovljivo evidenco. Ohranil sem izpise 948 lokalnih testov, strukturirane rezultate transakcijskih in podpisnih preizkusov ter dejanske zaslonske slike. Pri tem sem odstranil dostopne podatke in jasno navedel testni način. S takšnim gradivom lahko svoje ugotovitve pojasnim, preverim in uporabim kot osnovo za ponovni preizkus po spremembah programske opreme. Za nadaljnje delo bi najprej uskladil cenovni prikaz ter izboljšal obravnavo nalaganja plačilnega elementa. Nato bi ponovno izvedel celoten uporabniški scenarij in vključil več uporabnikov. S tem bi strokovni pregled dopolnil z meritvijo dejanskega razumevanja in uspešnosti nalog. Pri zaledju bi razširil preizkuse na nadzorovane prekinitve posameznih korakov ter še naprej preverjal nespremenljivost poslovnih učinkov ob ponovitvah.

Pri pripravi diplomskega dela sem ugotovil, da kakovost integracije najlažje presodim na stikih med sistemi. Na teh mestih se odločim, kateri ceni zaupam, kateri dogodek sprejmem, kdaj ustvarim rezervacijo in kaj povem uporabniku. React in Next.js mi ponujata različni možnosti organizacije teh stikov, odgovornosti pa ostanejo moje. Kot osrednji sklep zato izpostavljam potrebo po skupnem, preverjenem poslovnem stanju in uporabniškem prikazu, ki to stanje jasno ter pravilno predstavi.

7 VIRI IN LITERATURA

Karničar, A. (2026). Zapisniki izvedenih testov in posnetki zaslona z dne 16. 9. 2026. Lastno raziskovalno gradivo v elektronski prilogi.

Next.js. (b. d.-a). *Environment Variables*. Pridobljeno 16. 9. 2026 iz Next.js Documentation: https://nextjs.org/docs/app/guides/environment-variables

Next.js. (b. d.-b). *Route Handlers*. Pridobljeno 16. 9. 2026 iz Next.js Documentation: https://nextjs.org/docs/app/getting-started/route-handlers

Next.js. (b. d.-c). *Server Actions*. Pridobljeno 16. 9. 2026 iz Next.js Documentation: https://nextjs.org/docs/app/guides/server-actions

Nielsen, J. (1994). *10 Usability Heuristics for User Interface Design*. Pridobljeno 16. 9. 2026 iz Nielsen Norman Group: https://www.nngroup.com/articles/ten-usability-heuristics/

React. (b. d.). *Quick Start*. Pridobljeno 16. 9. 2026 iz React Documentation: https://react.dev/learn

Stripe. (b. d.-a). *Checkout Sessions*. Pridobljeno 16. 9. 2026 iz Stripe Documentation: https://docs.stripe.com/api/checkout/sessions

Stripe. (b. d.-b). *The Payment Intents API*. Pridobljeno 16. 9. 2026 iz Stripe Documentation: https://docs.stripe.com/payments/payment-intents

Stripe. (b. d.-c). *Receive Stripe events in your webhook endpoint*. Pridobljeno 16. 9. 2026 iz Stripe Documentation: https://docs.stripe.com/webhooks

Stripe. (b. d.-d). *Resolve webhook signature verification errors*. Pridobljeno 16. 9. 2026 iz Stripe Documentation: https://docs.stripe.com/webhooks/signature

Stripe. (b. d.-e). *Testing*. Pridobljeno 16. 9. 2026 iz Stripe Documentation: https://docs.stripe.com/testing

Supabase. (b. d.-a). *Edge Functions*. Pridobljeno 16. 9. 2026 iz Supabase Documentation: https://supabase.com/docs/guides/functions

Supabase. (b. d.-b). *Environment Variables*. Pridobljeno 16. 9. 2026 iz Supabase Documentation: https://supabase.com/docs/guides/functions/secrets

World Wide Web Consortium [W3C]. (12. december 2024). *Web Content Accessibility Guidelines (WCAG) 2.2*. Pridobljeno 16. 9 2026 iz W3C Recommendation: https://www.w3.org/TR/2024/REC-WCAG22-20241212/

Zeen Connect. (2026). Izvorna koda projekta ter testni skripti stripe-real-staging-e2e.mjs in staging-webhook-smoke.mjs. Interno razvojno gradivo, pregledano 16. 9. 2026; izbrane datoteke vključim v elektronsko prilogo.