VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ KATEDRA

FoodKept

KETVIRTASIS LABORATORINIS DARBAS

Programų sistemų inžinerija II

Darbą atliko: Programų sistemų II kurso studentai
Andrius Tumšys,
Arvydas Venskus,
Danielius Rėkus,
Ema Sinkevičiūtė

Vilnius,
2022

Santrauka

Šio darbo tikslas apibrėžti tobulinamos sistemos architektūrą naudodami pjūvius bei perspektyvas. Jų pagalba siekiame tiek įvertinti ir išanalizuoti pakeitimą, tiek implementuoti automatizuotus testus. Šios užduoties pagrindiniai tikslai:

1. Apibrėžti sistemos architektūrą, pakeitimą ei nefunkcinius reikalavimus atsižvelgiant į architektūrinius stilius.
2. Pateikti pjūvius bei perspektyvas bei argumentuoti, nurodyti motyvus, skatinusius vaizduoti pasirinktą architektūrą.
3. Implementuoti bent dalį pakeitimų
4. Implementuoti automatizuotą integravimą arba sistemos testą, integruoti ji su CI/CD procesais.

Taigi, šis darbas bus sudarytas iš konteksto, pjūvių, testavimo detalių bei atsekamumo matricos.

Turinys

1. Reikalavimai

Norėdami paaiškinti ir aiškiau aprašyti išsikeltus reikalavimus, sukūrėme reikalavimų sąrašą.:

1. Sukurti atskirą puslapį (arba papildyti esamą), kuriame pirkėjai galėtų nusipirkti produktą, bei matytų užsakymo statusą.

1.1 Puslapis turi atvaizduoti informaciją apie prekės įsigijimą:

1.1.1 Pardavėjo pavadinimą

1.1.2 Pardavėjo adresą

1.1.3 Prekės/produkto nuotrauką

1.1.4 Prekės/produkto kainą

1.1.5 Prekės/produkto aprašymą

1.1.6 Užsakymo apmokėjimo tipą:

1.1.6.1 Apmokėti dabar - pavedimu arba per atskirą mokėjimo sistemą

1.1.6.2 Apmokėti atsiėmus prekę

1.1.7 Užsakymo būsenos indikatorių (laukiama apmokėjimo/galima atsiimti/ruošiama)

1.2 Puslapyje turi būti pasirinkimas atšaukti užsakymą (tam tikrai atvejais, daugiau – 3.1)

2. Sukurti puslapio alternatyvą pardavėjams.

2.1 Puslapis turi atvaizduoti informaciją apie užsakymą:

2.1.1 Pirkėjo vardą

2.1.2 Prekės pavadinimą bei kiekį

2.1.3 Užsakymo kainą

2.1.4 Pirkėjo pasirinktą apmokėjimo būdą bei būseną

2.2 Puslapyje turi būti suteikta galimybė keisti užsakymo būsenos indikatorių (laukiama apmokėjimo/galima atsiimti/ruošiama)

2. Pjūviai

Yra daug skirtingų architektūros pjūvių, į kuriuos reikia atsižvelgti analizuojant programinės įrangos sistemą, tačiau ne visi jie yra svarbūs kiekvienai sistemai. Pasirinkome išanalizuoti pjūvius, kurie, mūsų nuomone, šiuo metu yra tinkamiausi ir aktualiausi mūsų projektui.

2.1 Konteksto pjūvis

Konteksto pjūvis – vienas pagrindinių, jis apibūdina ryšius, priklausomybes ir sąveiką tarp sistemos ir jos aplinkos. Tai padės pristatyti, kokie pakeitimai buvo įdiegti sistemoje ir apibrėžti, kuriose sistemos vietose jie diegiami.

2.1.1 Sistemos apimtis ir atsakomybės

Mūsų sistema jungia maisto prekių įmones su klientų baze. Pagrindinis privalumas vartotojams, jog klientai gali rasti maisto prekių, kurios dažniausiai yra ekologiškos, ir jas pirkdami sumažina maisto eikvojimą. Sistema padeda užmegzti ryšį tarp kliento ir pardavėjo suteikdama naudos abejoms pusėms. Paslauga yra nemokama naudojimui. Pakeitimas suteiks galimybę klientams atsiskaityti už prekes.

2.1.2 Išorinės esybės, paslaugos ir naudojami duomenys

Mūsų internetinė aplikacija jungia pardavėjus (restoranus, maitinimo įstaigas, turinčias maisto likučių, kuriuos nori pigiau parduoti) bei pirkėjus (fizinius asmenis, norinčius įsigyti produktų pigiau bei prisidėti prie gamtos išteklių tausojimo). Tam, kad sistema sėkmingai funkcionuotų, numatome ne vien išorinių esybių panaudojimą produktų valdymo bei paieškos etapuose, bet ir išorinių paslaugų naudojimą apmokėjimo žingsnyje.

Atsiskaitant už prekes ir žaliavas, pirkėjai nukreipiami į atskiras sistemas (PaySera, PayPal ir kt.), iš kur mūsų sistemos pardavėjai gauna informaciją apie užsakymo apmokėjimą. Tokiu būdu sistema užtikrina patikimą ir saugų apmokėjimo būdą bei informuoja tiek pirkėją, tiek pardavėją apie įvykdytą mokėjimą.

2.1.3 Išorinių esybių pobūdis ir charakteristikos

Mūsų aplikacija atsiskaitymo metu keičiasi informacija su kitomis sistemomis, esančiomis įvairiose lokacijose. Kai kurios iš šių sistemų dėl tam tikrų taisymų ar pokyčių gali

būti prieinamos tik tam tikru metu. Tačiau nesėkmingas ryšys su tokia sistema gali sukelti kliento užsakymo praradimą. Todėl mūsų sistemos sąsaja su išorinėmis sistemomis turi būti kruopščiai suprojektuota. Visos nesėkmingos sąsajos turėtų būti pakartotinai konfigūruojamos.

2.1.4 Išorinės sąsajos tipas ir paskirtis

Mūsų aplikacijos pirkėjas užsakymo apmokėjimo žingsnyje pasirinkus mokėjimą per išorines sistemas tokias kaip PaySera ar PayPal yra nukreipiamas į atitinkamus puslapius, kurios suvedus bankinius duomenis ir įvykdžius mokėjimą atsiunčia į FoodKept sistemą patvirtinimo pranešimą, kurio dėka pardavėjas mato apmokėjusio pirkėjo duomenis ir gali pavirtinti užsakymą. Aprašytas funkcionalumas leidžia vadinti šias sistemas tiek duomenų, tiek paslaugų tiekėjomis mūsų sistemos atžvilgiu.

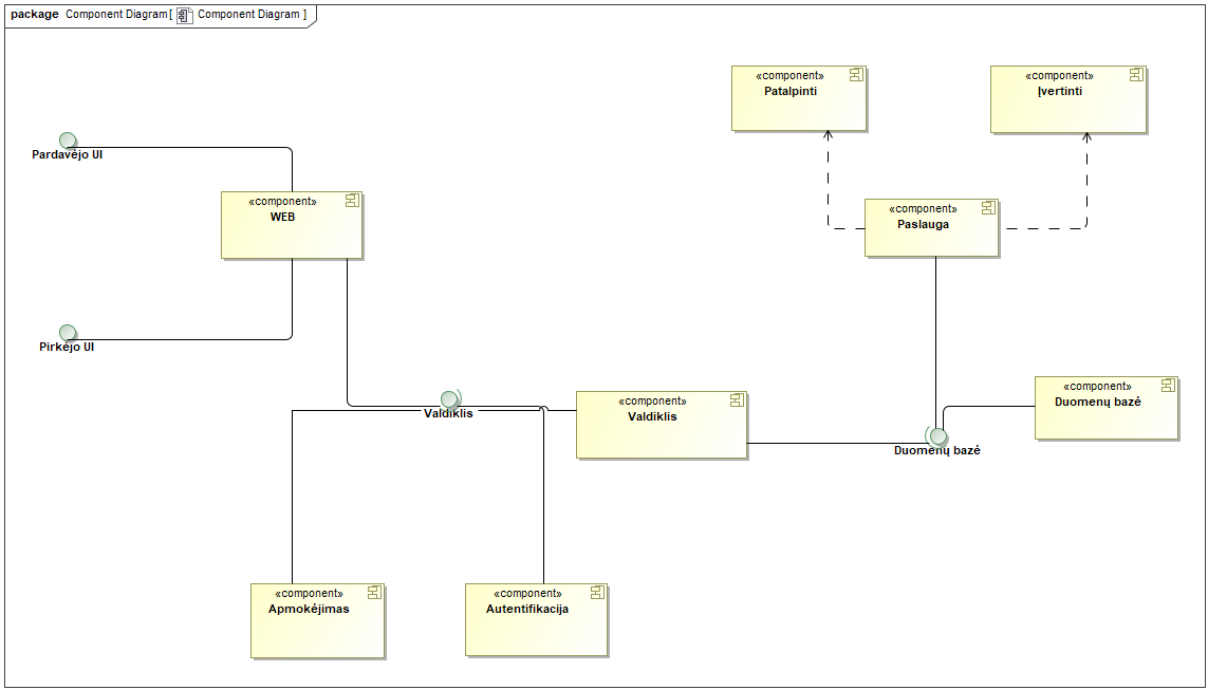
2.1.5 Kitos išorinės priklausomybės

Kadangi mūsų sistemos numatyta, kad vienas iš apmokėjimų būdų yra atsiskaitymas per išorines organizacijas, kurios vykdo mokėjimus bei susieja pirkėjo banko sąskaitą su savo paskyromis, galima teigti, jog jų vidiniai procesai irgi yra tarpusavyje priklausomi – išorinėje sistemoje privalo būti pateikti bei patvirtinti korektiški duomenys, nurodant asmens bei kitus autorizacijos duomenis.

2.1.6 Bendras nuoseklumas ir rišlumas

Mūsų sistemos tikslas sujungti pirkėjus bei pardavėjus ir tuo pat sumažinti maisto švaistymą, pirminė sistemos versija užtikrino akį traukiantį dizainą ir pagrindinius produktų įkėlimo bei paieškos funkcionalumus, tačiau buvo kiek nepatogi vartotojams bei pardavėjams nepateikdama atsiskaitymo už prekes bei užsakymo valdymo būdo. Papildydami sistemą minėtais pakeitimais ir naudodami išorines sistemas siekiame padidinti klientų pasitenkinimą bei programos populiarumą.

2.1.7 UML komponentų diagrama



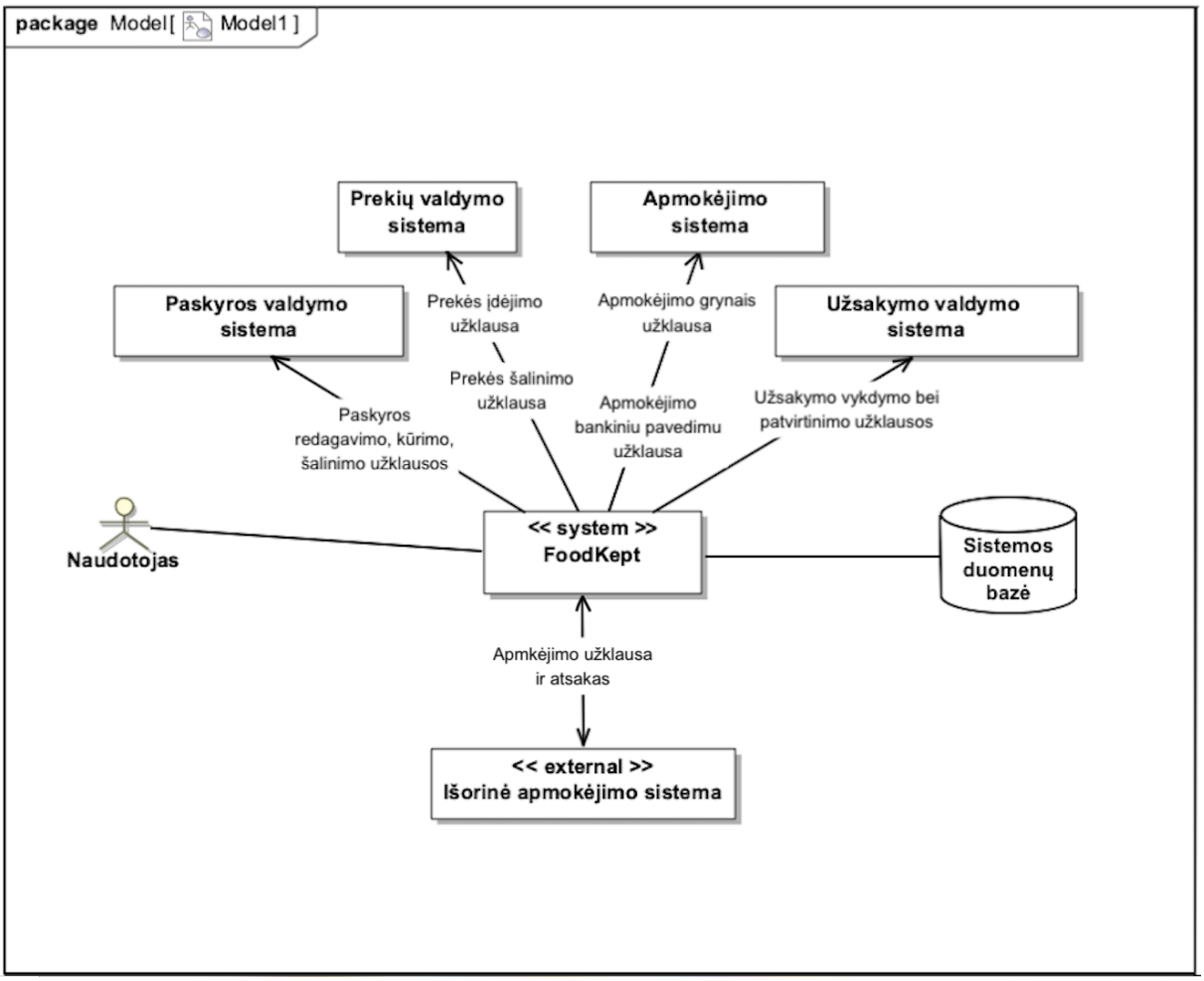
Pav. 1 Komponentų diagrama

Šiame skyriuje vaizduojamas sistemos implementavimas ir komponentai. Kiekvieną apmokėjimą sukonstruoja valdiklis naudodamas apmokėjimo duomenis iš duomenų bazės. Šie duomenys saugomi duomenų bazėje. Tai pavaizduota 1 pav.

2.1.8 UML Konteksto Diagramos remiantis panaudos atvejų modeliu

Naujasis pakeitimas praplečia pirkėjų galimybę patogiau atsiskaityti už prekes bei žaliavas, o pardavėjams suteikia galimybę sekti užsakymo būsenas bei jas tvirtinti. Pav. 2 yra paruoštas pagal Nick Rozanski ir Edin Woods knygoje “Software Systems Architecture” pateiktas gaires. Diagrama remiasi UML standartu, kuris padeda atvaizduoti sistemos po pakeitimo dalių tarpusavio sąveiką, tačiau yra papildyta :

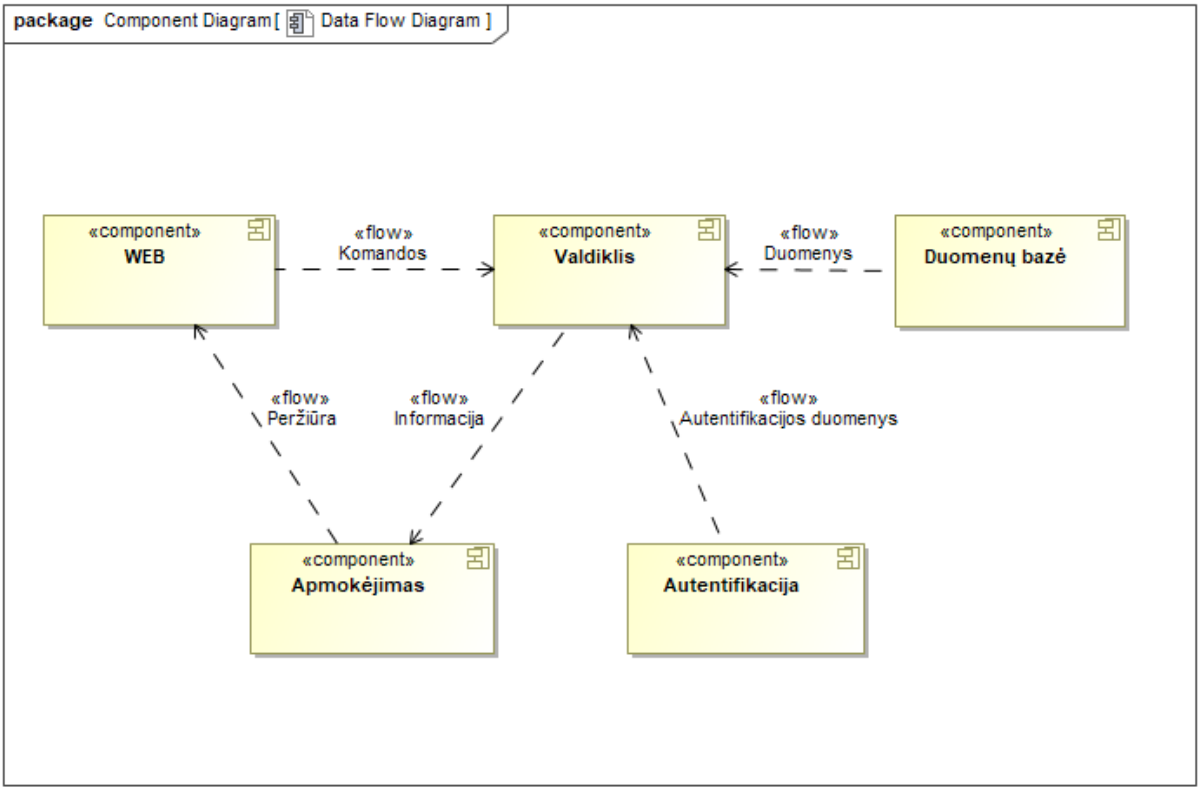
- Aktorius “Naudotojas” atvaizduoja išorines esybes, sąveikaujančias su sistema tokias kaip Pirkėjas bei Pardavėjas.
- Pati sistema vaizduojama kaip komponentas su subsystem stereotipu, o išorinei sistemai pavaizduoti naudojamas komponentas su external stereotipu
- Sąsajos tarp vidinių ir išorinių sistemos dalių atvaizduotos kaip informacijos srautas iš vienos sistemos į kitą.



Pav. 2 panaudos atvejų pagrindu sudaryta konteksto diagrama

2.1.9 Informacijos Srauto Diagrama

3 pav. vaizduojama informacijos srauto diagrama rodo, kaip perduodama informacija tarp komponentų. Apmokėjimo metu gaunama informaciją iš valdiklio, kuris gauna duomenis iš duomenų bazės. Vartotojas per žiniatinklio komponentą gali matyti apmokėjimo galimybę ir nuspręsti, ką daryti toliau. Žiniatinklio komponentas siunčia komandas valdiklio komponentui, ką vartotojas nori daryti toliau.



Pav. 3 Informacijos Srauto Diagrama

2.1.10 Sąveikos scenarijai

Įdiegus pakeitimą pirkėjai galės apmokėti užsakymą

1. Bankiniu pavedimu:

   Galimi scenarijai:

   a) Sėkmingai atliktas bankinis pavedimas, pardavėjas gavęs pinigus paspaudžia užsakymo lange patvirtinimo mygtuką ir sistema įgalina jį redaguoti/atšaukti užsakymą.

   b) Per 15 min nepavykus apmokėti užsakymo, rezervacija atšaukiama, prekės, buvusios neapmokėtame krepšelyje grąžinamos į internetinę parduotuvę.

2. Mokėjimas grynais atsiimant

Vienintelis scenarijus: prekės perkeliamos į pardavėjo vykdomų užsakymų sąrašą ir pardavėjas arba atšaukia užsakymą pirkėjui neatvykus, arba pažymi jį kaip įvykdytą pirkėjui apmokėjus ir atvykus atsiimti prekes.

3. Per išorines organizacijas
   Sistema sąveikauja su išorine mokėjimų sistema ir nukreipia pirkėją į atitinkamo adreso svetainę, toliau sistema gauna pranešimą iš išorinės organizacijos:
   a) Išorinė sistema atsiunčia sėkmingo apmokėjimo pranešimą – sistema rezervuoja užsakyme esančias prekes ir pakeičia jų statusą į apmokėtą, pardavėjai iškart mato užsakymo būseną.
   b) Išorinė sistema atsiunčia nesėkmingo apmokėjimo pranešimą – sistema atšaukia prekių rezervacijas ir jos grįžta į elektroninę parduotuvę.

2.1.11 Konteksto pjūvio perspektyvos

2.1.11.1 Saugumo perspektyva

Sistema pasiekiama vartotojams per žiniatinklio vartotojo sąsają. Tai sukelia DDoS ir DoS atakų grėsmę. Tuo pačiu principu atakų gresmė gali būti sukeliama norint pasiekti paskyras naudojant „brute-force“. Daugumą šių problemų būtų galima išspręsti pasirinkus hostingo paslaugas, kurios užtikrina apsaugą nuo minėtų problemų.

2.1.11.2 Greitaveikos ir plėtros perspektyva

FoodKept internetinėje aplikacijoje siekiame užtikrinti kokybišką ir greitą sistemos veikimą, todėl išsikeliame pagrindinį sistemos greitaveikos reikalavimą ir aprašome plėtros perspektyvą:

a) Kiekvienas naudotojo sąveikos elementas privalo grąžinti pilną rezultatą paspaudus mygtuką per ilgiausiai 3 sekundes kai internetinės svetainės apkrova ne didesnė nei 100 naudotojų vienu metu.
b) Kadangi sistema sukurta kaip akademinis projektas pirminiai apkrovos reikalavimai yra šimtui naudotojų sėkmingai naudotis sistema neviršijant 3 sekundžių užsikrovimo laiko.

2.1.11.3 Prieinamumo ir atsparumo perspektyva

Mūsų aplikacija apmokėjimo metu naudoja trečių šalių apmokėjimo galimybę. Tinklas tarp aplikacijos ir trečių šalių gali nutrūkti, todėl gali būti išjungtas techninei priežiūrai. Kai tinklas yra nepasiekiamas arba nereaguoja, apmokėjimas negali būti įvykdytas. Todėl apmokėjimo metu pirkėjas galės pasirinkti apmokėti tik kitais būdais.

2.1.11.4 Evoliucijos perspektyva

Sistemoje galimi funkciniai pakeitimai kode, MVC modelis leidžia pasiekti ir koreguoti reikiamas kodo vietas. Kadangi sistema sukurta kaip internetinė aplikacija, ji nesukelia sunkumų migruojant iš vienos platformos į kitą. Tai, kad jau esame implementavę API pardavėjų produktams įkelti, parodo, jog sistema yra palanki integracijos pokyčiams. Galiausiai, atsižvelgiant į sistemos perspektyvą augti turime užtikrinti galimybę didinti vartotojų srautą neprarasdami sistemos greitaveikos, ką stengiamės realizuoti naudodami API.

2.2 Funkcionalumo pjūvis

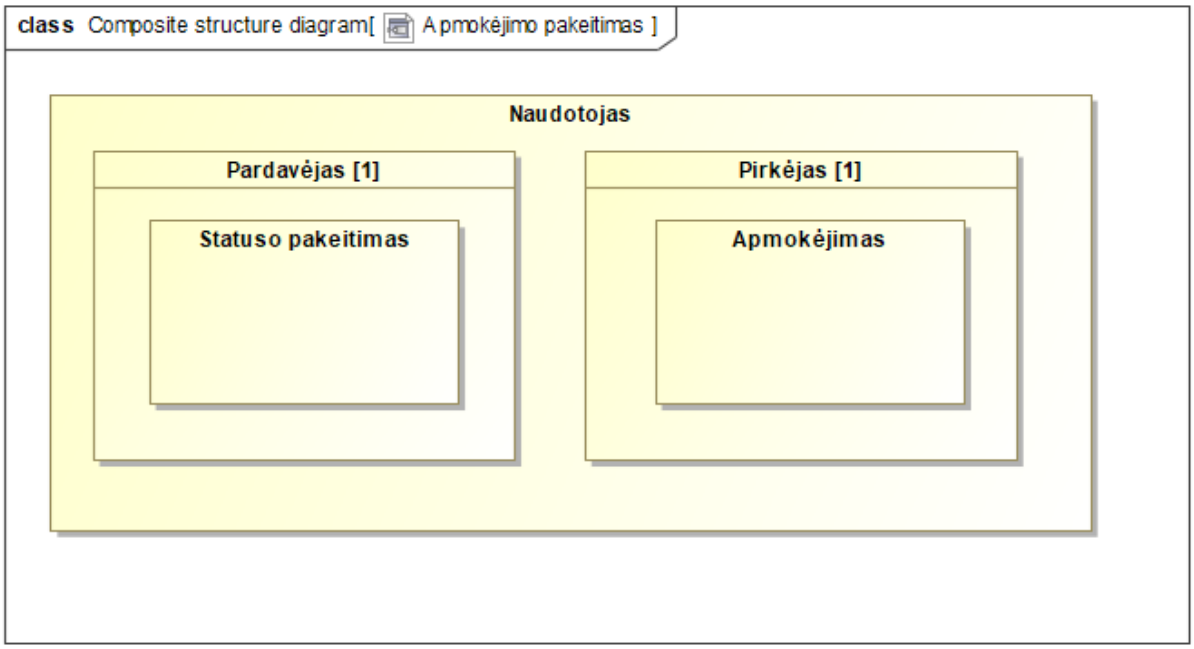
Funkcionalumo pjūvis aprašo sistemos funkcinius elementus, jų tikslus, sąveiką, bei parodo, kaip sistema realizuoja suinteresuotų asmenų iškeltus funkcinius reikalavimus. Šį pjūvį pasirinkome su tikslu parodyti, jog sistemos pakeitimai tinka prie dabartinio projekto architektūros ir nekeisdami jos tipo, gražiai ją papildo.

2.2.1 Dabartinis Funkcionalumas

Šiuo metu, didelę dalį sistemos funkcionalumo sudaro duombazė. Visa pardavėjų, pirkėjų, produktų ir kita informacija yra surenkama puslapyje ir išsaugoma duomazėje. Būtent todėl sistemos funkcionalumas gali būti aprašomas, kaip ji visą duombazės informaciją apdoroja, filtruoja, rodo naudotojams ar ją keičia. Dėl šios priežasties mūsų sistema yra MVC (models, views, controllers) architektūros tipo. Šie sistemos komponentai atlieka visą darbą, kurio dėka vartotojai gauna norimą informaciją.

2.2.2 UML Grupinė struktūros diagrama

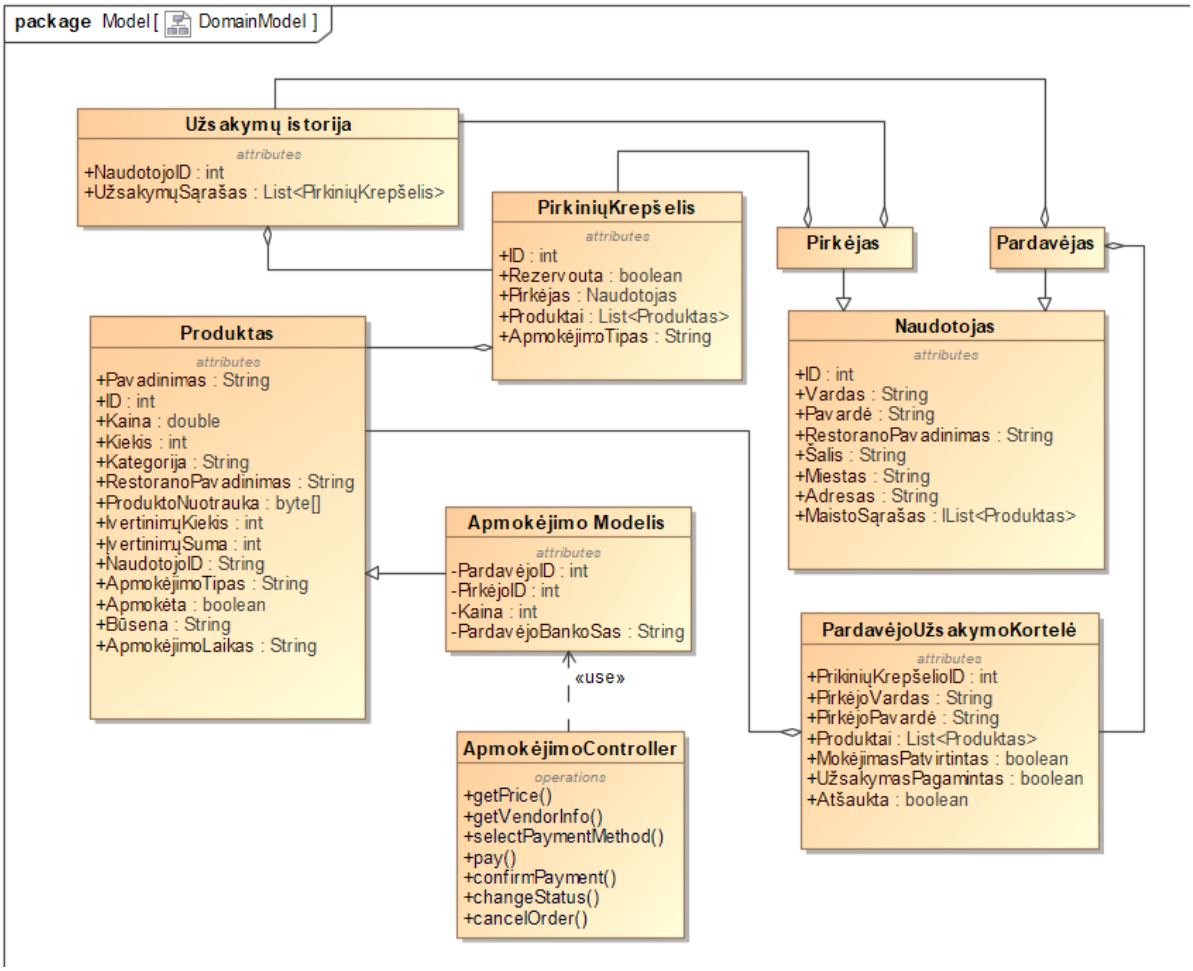
Ši diagrama vaizduoja, kaip mūsų apmokėjimo pakeitimas atrodys sistemos struktūroje. Kaip matoma, ji susies vartotojų klases “pirkėjas” ir “pardavėjas”, leisdama pirkėjui apmokėti užsakymą, o pardavėjui keisti jo būseną.



Pav. 4 Grupinė struktūros diagrama

2.2.3 Pakeitimo implementacija

Naujas pakeitimas, kaip ir rodo klasių diagrama, susies pardavėjus su pirkėjais, leisdamas pirkėjams pasirinkti kaip ir kada atsiskaityti už prekes, o pardavėjams atvaizduoti kuo tikslesnę informaciją keičiant užsakymo būseną. Šis pakeitimas bus implementuotas sekant tą pačia MVC sistemos architektūrą, pridedant Apmokėjimo modelio ir ApmokėjimoController klases, kurios praplės vartotojų galimybes nekeičiant architektūros, o tik papildant sistemos funkcionalumą.



Pav. 5 Klasių diagrama

2.2.4 UML Komponentų diagrama

Kaip parodyta pav.1, sistema atlieka savo uždavinius “database-centric” architektūros modeliu. Funkcionalumas yra darbas su duomenų baze – informacijos įkėlimas, pildymas, keitimas ar atvaizdavimas vartotojams per valdiklį, todėl ir po pakeitimo, sistemos architektūra išliks tokio pačio tipo, bei veikimo principo.

2.2.5 Evoliucijos perspektyva

Sistemos funkcionalumo evoliucijos perspektyvą jau šiek tiek aptarta 2.1.11.4 skyriuje. Nuo MVC modelio migruoti būtų per sunku ir neverta, todėl didžioji sistemos dalis nesikeis, nebent tobulės valdiklių greitaveika, pridedant gijas, SQL injekcijas (kur jų dar nėra). Duomenų bazė patobulės tik pakeitus informacijos saugojimą į 4nf. Tokiu būdu informacija nesidubliuos, nebus prarandama, nevyks kitos duomenų anomalijos.

2.2.6 Saugumo perspektyva

Kaip ir minėjome viršuje, jautriausias sistemos komponentas yra duombazė, kadangi joje laikoma visa mūsų turima informacija. Duombazės saugumas šiek tiek aprašomas 2.3.2 sekcijoje. Po pakeitimo duombazėje atsiras daugiau jautrios informacijos - pirkėjų ir pardavėjų vardai, pavardės, banko sąskaitos. Nors su šia informacija padaryti beveik nieko negalima, ji vistiek turi būti saugoma nuo pašalinių asmenų. Kiti sistemos komponentai, susiję su pakeitimu, naudojami tik informacijai skaityti iš duombazės, bei ją atvaizduoti pirkėjams.

2.2.7 Greitaveikos ir plėtros perspektyva

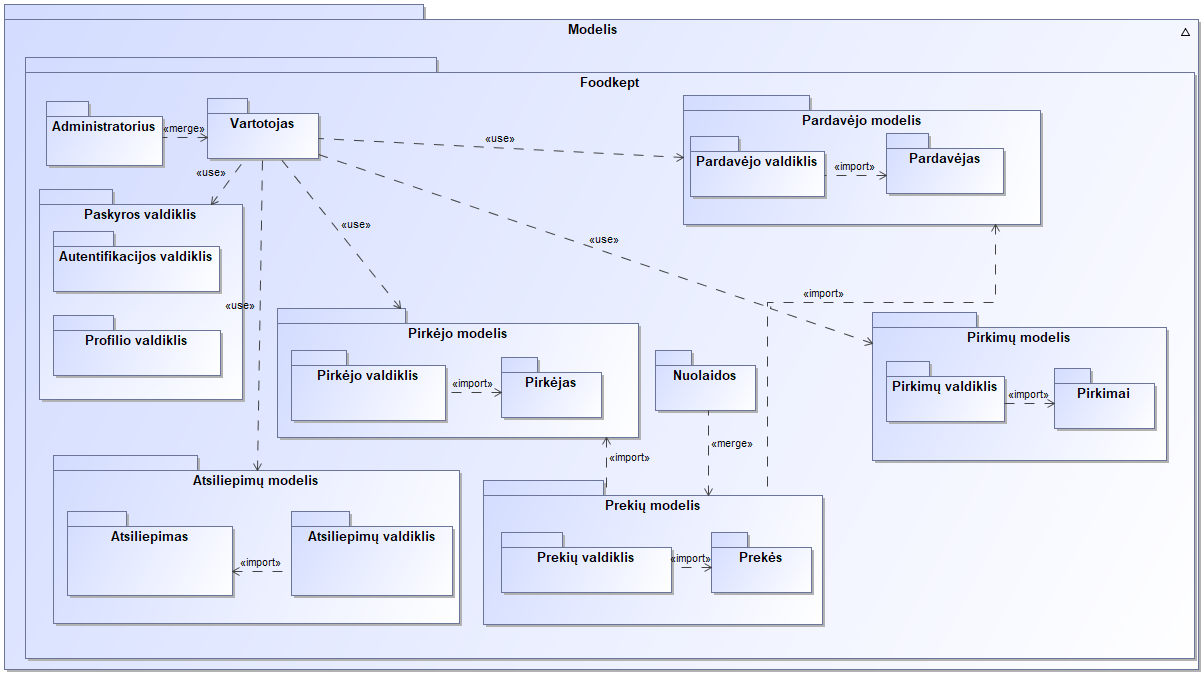
Norint pagerinti sistemos veikimą ir pagreitinti informacijos gavimo greitį, informacija duombazės lentelėse turėtų būti saugoma Ketvirtorios Normalinės formos (4nf) būdu. Naudojant 4nf lenteles, informacija duombazėje nesidubliuoja ir nevyksta duomenų anomalijos, tačiau taip pat pagreitėja ir informacijos paieška.

2.3 Programos vystymo pjūvis

Programos vystymo pjūvis nusako architektūrą kuri palaiko programos kūrimo ir vystymo procesą. Šis pjūvis mums padeda organizuoti ir standartizuoti visą projektą.

2.3.1 UML Paketų diagrama

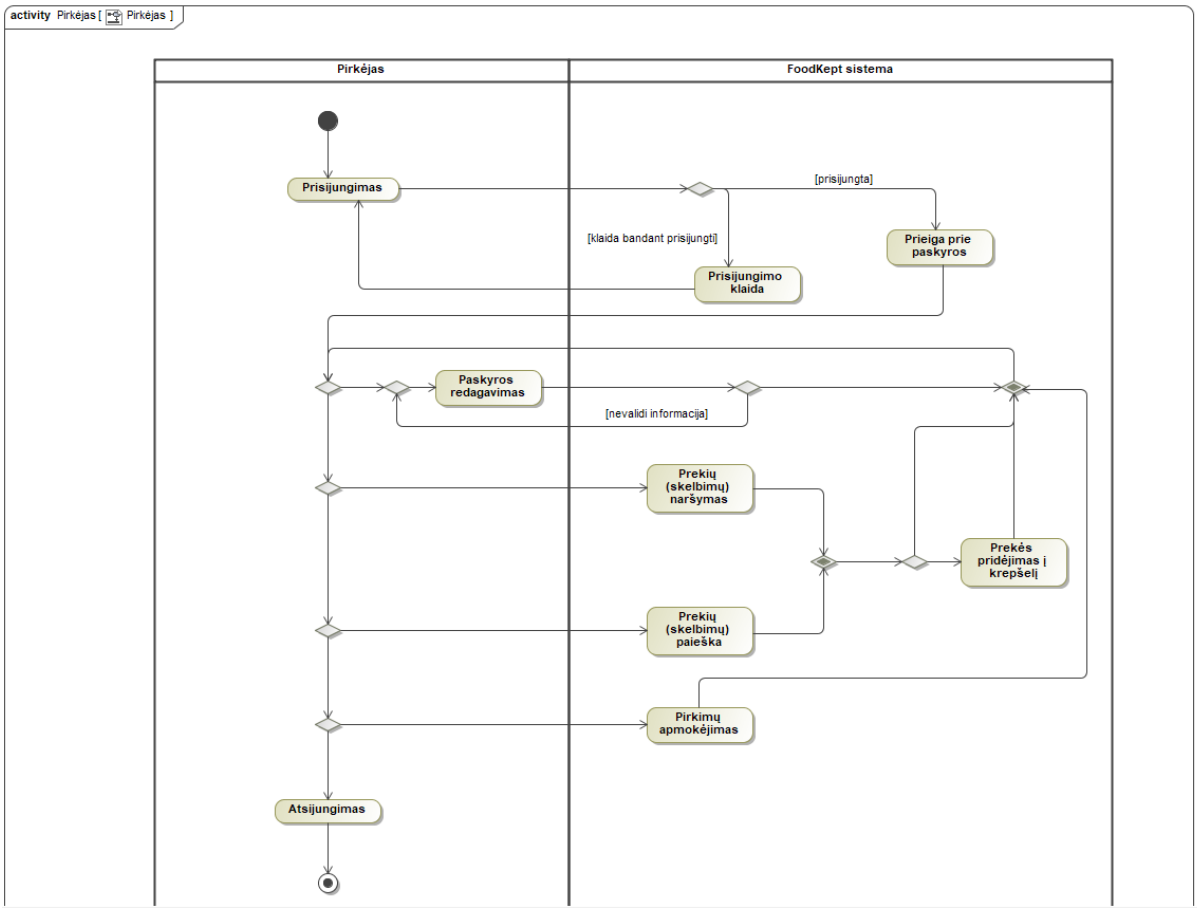
Mūsų pasirinktas MVC modelis puikiai tinka tokio tipo programai. Tai galime pamatyti pav. 6. Taip pat matome, kad pakeitimas nesukels papildomų rūpesčių jį įterpiant į bendrą programos architektūrą. Pridėtas pirkimų modelis yra suprojektuotas lygiai taip pat kaip ir kiti mūsų programos modeliai.



Pav. 6 Paketų diagrama

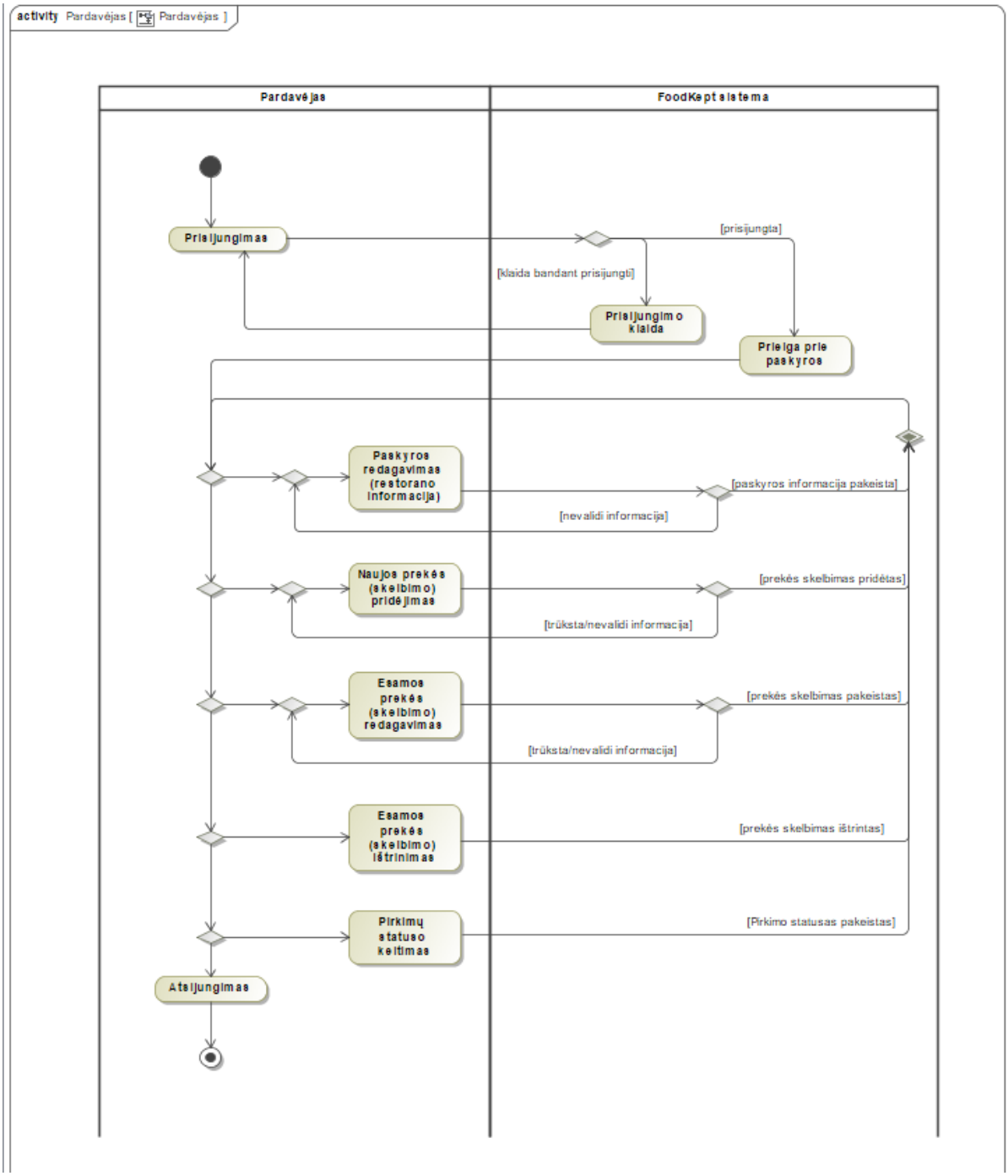
2.3.2 UML Veiklos diagrama

Mūsų pakeitimas taip pat tinka sistemos – kliento komunikavimo stiliui. Pirkimų vykdymas vyksta po prekių pridėjimo į krepšelį, todėl yra lengvai pridedamas elementas. Tai galime pamatyti pav. 7. Po pakeitimų pirkėjai galės ne tik įsidėti prekes į krepšelį, bet ir už jas sumokėti.



Pav. 7 Pirkėjo veiklos diagrama

Taip pat keisis ir pardavėjo vykdomi procesai, nes nuo šiol jis galės patvirtinti apmokėtus užsakymus ir keisti jų statusą. Šis pakeitimas taip pat lengvai pridedamas kaip papildomi veiksmai pardavėjui, kurie neturės įtakos buvusių procesų veikimui. Šiuos pakeitimus galime matyti pav 8.



Pav. 8 Pardavėjo veiklos diagrama

2.3.3 Kodo organizavimas

Versijų kontrolė yra užtikrinama naudojant sistemą Git. Mes naudojame GitHub valdyti Git repozitorijoms. Vykdant programos plėtrą yra naudojamas nenutrūkstamos integracijos

modelis, kuris užtikrina teisingą pakeitimų veikimą. Naudojantis šia sistema taip pat valdome ir programos konfigūraciją.

2.3.4 Bendrieji procesai

Sistemos veikla yra stebima ir kiekviena klaida yra užrašoma į žurnalą. Taip pat sistemai pažįstamos klaidos yra apdorojamos ir parodomos vartotojams suprantamu tekstu.

2.3.5 Greitaveikos ir plėtros perspektyva

Pakeitimas nepaveiks dabartinės greitaveikos, tačiau turime užtikrinti greitą veikimą integruojantis su trečiųjų šalių programomis. Vystydami šią sistemą turime užtikrinti, kad papildomų trečiųjų šalių programų prijungimas reikalautų tik konfigūracijos pakeitimų, tačiau veikimas išliktų identiškas.

2.4 Informacinis pjūvis

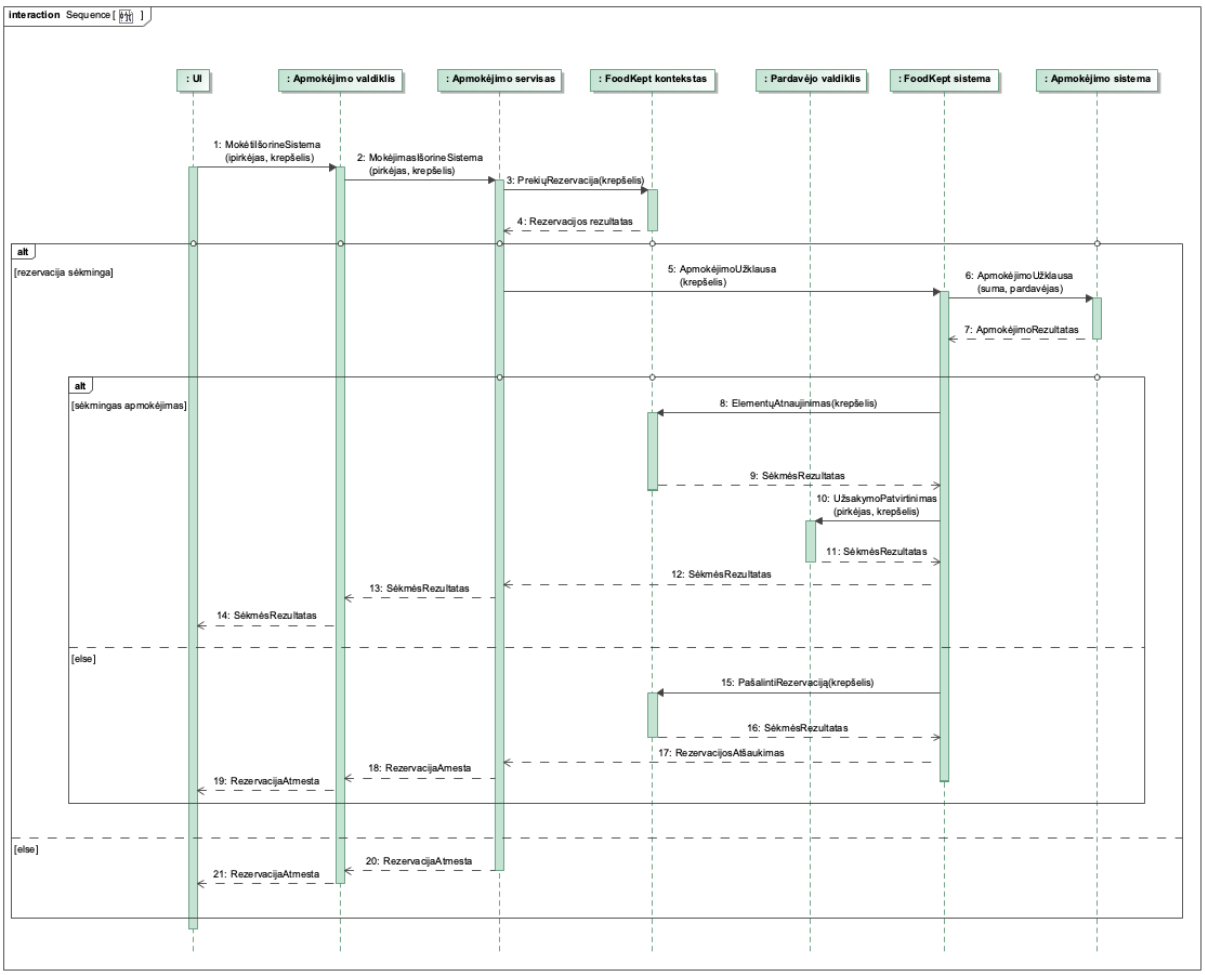
Informaciniame pjūvyje siekiame išnagrinėti sistemos informacijos srautus bei su jais susijusius procesus ar struktūrą pagal duomenų tipus arba informacijos elementus.

2.4.1 Informacijos srauto diagrama

Informacijos srauto diagrama rodo, kaip perduodama informacija tarp komponentų. Apmokėjimo metu gaunama informaciją iš valdiklio, kuris gauna duomenis iš duomenų bazės. Vartotojas per žiniatinklio komponentą gali matyti apmokėjimo galimybę ir nuspręsti, ką daryti toliau. Žiniatinklio komponentas siunčia komandas valdiklio komponentui, ką vartotojas nori daryti toliau. Informacija perduodama iš duomenų bazės. Diagrama pavaizduota 3 pav.

2.4.2 Informacijos sekų diagrama

Siekdami tiksliau atvaizduoti informacijos perdavimą sistemoje nubrėžėme apmokėjimo proceso sekų diagramą. Dėl aiškumo vaizduojame tik apmokėjimo išorinėmis sistemomis procesą, mokėjimas bakiniu pavedimu duomenų atžvilgiu yra analogiškas, skiriasi vien tuo, jog sistema nesiunčia užklausos bankui, o gauna patvirtinimą ir atitinkamai siunčia pranešimą pardavėjui. Mokėjimas grynaisiais yra dar paprastesnis, nes jį sudaro mažiausiai duomenų perdavimo ir modifikavimo operacijų. Įvykdžius užsakymą prekės rezervuojamos, o patvirtinimo pranešimo nelaukiama.



Pav 9. Apmokėjimo sekų diagrama

2.4.3 Informacijos struktūra ir turinys

Vartotojo duomenys saugomi duomenų bazėje, kurie per valdiklį perduodami į apmokėjimą. Pardavėjas gali matyti vartotojo pirkinių krepšelį, kuris taip pat perduodamas analogiškai kaip ir vartotojo duomenys į apmokėjimą.

2.4.4 Informacijos paskirtis ir panaudojimas

Dėl nedidelio projekto mąsto, visi duomenys yra saugomi vienoje duomenų bazėje, nekuriant transakcijų ar ataskaitos duomenų bazių. Toks sprendimas aplikacijai pasiekus didelį naudotojų kiekį galėtų sukelti sunkumų, kadangi užklausų kiekis į tą pačią duomenų bazę labai išaugtų sudarydamas didelę apkrovą.

2.4.5 Informacijos nuosavybės teisė

Vartotojo duomenys saugomi duomenų bazėje, prie kurių prieigą turi tik patys paskyros vartotojai ir administratoriai, kurie kontroliuoja vartotojų prisijungimus ir įkeliamą turinį.

2.4.6 Įmonei priklausanti informacija

Prieiga prie vienos centrinės saugyklos, kuri yra valdoma mūsų sistemos užtikrina, kad duomenys atnaujinami kiekvieną kartą prisijungus, tačiau, kaip jau minėjome, saugykla tampa apkrauta augant naudotojų skaičiui ir gali įtakoti viekimo sutrikimų.

2.4.7 Identifikatoriai

Duomenų bazėje be to, kad vienas el. paštas gali turėti tik vieną paskyrą, kiekvienam naudotojui priskiriamas identifikacinis numeris, pagal kurį sistema suranda atitinkamus produktus, krepšelių turinius ir kitus veikimui reikalingus atributus. Tokiu būdu išvengiama paieškos pagal kelis raktus, kurie sudaro pirminį esybės raktą.

2.4.8 Informacijos saugojimo modeliai

Sistemoje naudojamas reliacinis duomenų bazės modelis, kurio norminės formos užtikrina duomenų vientisumą, efektyvų naudojimą, bei duomenų pertekliaus minimizavimą. Duomenims pasiekti naudojama aplikacijų programavimo sąsaja.

2.4.9 Informacijos Srautas

Informacijos srautas vaizduojamas aptartoje informacijos srauto diagramoje. Apmokėjimo metu gaunama informaciją iš valdiklio, kuris gauna duomenis iš duomenų bazės. Vartotojas per žiniatinklio komponentą gali matyti apmokėjimo galimybę ir nuspręsti, ką daryti toliau. Visa informacija perduodama iš duomenų bazės.

2.4.10 Informacijos nuoseklumas

Duomenų nuoseklumui palaikyti sistemoje būtina įvesti taisykles, kurios sukontroliuotų operacijų netikslumus skatinančias situacijas.

1. Kadangi esant dideliam pirkėjų skaičiui keli pirkėjai gali įsidėti į krepšelį tą pačią prekę, būtina patikrinti, ar į pirkinių krepšelį įdėti pirkiniai vis dar yra prekyboje, nėra rezervuoti ir neleisti įsidėti pirkinių, kurių statusas „rezervuoti“.
2. Pardavėjų paskyros pakeitimai turi būti uždrausti, jei jų produktai laukia apmokėjimo arba yra rezervuoti. Tokiu būdu užtikrinamas informacijos vienareikšmiškumas pirkėjo krepšelyje.
3. Produktų kainos negali būti keičiamos po įkėlimo – išvengiama konfliktų apmokėjimo metu.

3.1.1 Informacijos Kokybė

Informacijos kokybės valdymui įdiegėme administratorių, kuris reguliuoja tinkamą turinį. Naujas pakeitimas leis užtikrinti pirkinių krepšelių sudarymo kokybę, kadangi pardavėjas apmokėjimo skiltyje galės patvirtinti esamų produktų kiekį, taip apsaugodamas nuo pakartotinių užsakymų. Vartotojo informacijos duomenų kokybei užtikrinti naudojama registracija, kuri yra validuojama.

3.1.2 Savalaikiškumas, delsa, amžius

Kadangi visi duomenys saugomi vienoje duomenų bazėje ir pasiekiami realiu laiku sinchroniškai, savalaikiškumo ir duomenų amžiaus rodikliai geri, tačiau esant didelei apkrovai delsa gali padidėti, nes sistema turės apdoroti daug užklausų.

3.1.3 Archyvavimas ir informacijos saugojimas

Šiuo metu sistemoje nėra archyvavimo ir duomenų išsaugojimo juos praradus galimybės, todėl norint toliau ją plėtoti būtina tai apmąstyti. Tai užtikrintų sistemos patikimumą ir transakcijų atsekamumą.

3.1.4 Informacijos pjūvio perspektyvos

3.1.4.1 Saugumo perspektyva

Kaip ir minėjome viršuje, jautriausias sistemos komponentas yra duomenų bazė, kadangi joje laikoma visa mūsų turima informacija. Duomenų bazės saugumas aprašomas 2.3.2 ir 2.2.6. sekcijose. Duomenys turėtų būti prieinami tik autorizuotiems vartotojams ir administratoriui, kuris kontroliuoja tinkamą turinį.

3.1.4.2 Greitaveikos ir plėtros perspektyva

Kadangi naudojama viena duomenų bazė, į ją kreipiantis dideliam naudotojų skaičiui ji gali būti perkrauta, nuo ko nukentėtų greitaveika, sistema galėtų nustoti veikti. Taip pat žiūrint į plėtros perspektyvą būtina atsižvelgti į tai, jog geografiškai atokiai funkcionuojančios sitemos gali patirti greitaveikos problemų dėl pasaulinio tinklo (global network) apribojimų.

3.1.4.3 Prieinamumo perspektyva

Vartotojo duomenys privalo būti prienami tik prisijungusiam ar prisiregistravusiam vartotojam ir administratoriui. Pirkinių krepšelio duomenys bus prieinami ir pirkėjui, tačiau jie negali būti prieinami kitiems vartotojams.

3.1.4.4 Evoliucijos perspektyva

Plečiantis sistemai gali atsirasti poreikis apdoroti vis daugiau užklausų, turėti vieną centralizuotą duomenų bazę, kurios paprastas pigus, paprastas ir greitas implementavimas gali tapti problematiška. Toks architektūrinis sprendimas gali sumažinti dideliu atstumu nutolusių naudotojų greitaveiką, todėl svarbu apsvarstyti duomenų paskirstymo, lokalizavimo galimybes.

3.2 Lygiagretinimo (concurrency) pjūvis

Lygiagretinimo pjūvyje apibūdinama lygiagreti sistema ir priskiria funkcinius elementus, skirtus identifikuoti sistemą, kuri gali veikti vienu metu.

3.2.1 Lygiagretinimo pjūvio analizė

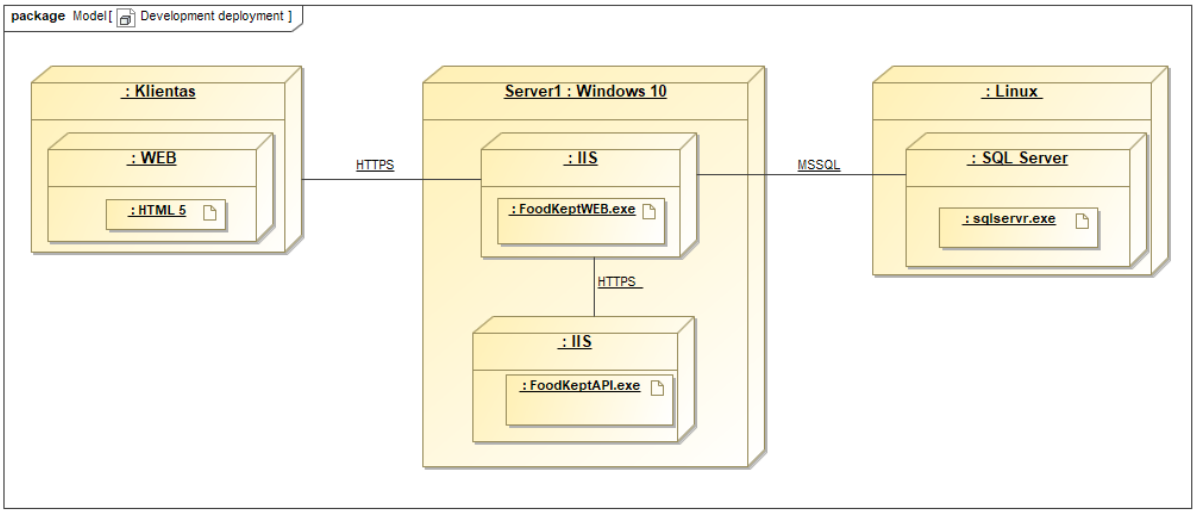
Keliose sistemose vietoje naudojamas lygiagretinimas gijomis (threads), tačiau jo tikslas yra greitinti sistemos veiklą. Kadangi užsakymo apmokėjimai įvyksta pakankamai greitai, be to dažnai yra naudojamos išorinės sistemos, šio pakeitimo metu apie lygiagretinimą galvoti nėra būtina.

3.3 Diegimo pjūvis

Diegimo pjūvyje dėmesys sutelkiamas į sistemos aspektus, kurie yra svarbūs po to, kai sistema buvo išbandyta ir paruošta pradėti veikti. Šiame pjūvyje apibrėžiama fizinė aplinka, kurioje sistema turi veikti įskaitant įrangos aplinką, kurios reikia sistemai.

3.3.1 Aplinkos Diegimo Diagrama

Šioje dalyje išanalizavome techninio lygmens programinės įrangos komponentų topologiją bei fizinius ryšius tarp šių komponentų. Sistemos vykdymo aplinkos parodytos 10 pav. Vartotojo įrenginio operacinė sistema nėra svarbi, nes visų tipų vartotojai prieis prie sistemos per naršyklę. Duomenų bazė nebūtinai turi būti įdiegta Linux vykdymo aplinkoje, tačiau tai rekomenduojama. Rekomenduojama, kad serverio valdiklių vykdymo aplinka būtų „Windows“ operacinė sistema, nes ji buvo sukurta naudojant ASP.NET sistemą.



Pav. 10 Aplinkos Diegimo Diagrama

3.3.2 Diegimo pjūvio analizė

Kadangi po pakeitimo diegimas išliks toks pat kaip ir anksčiau, šio pjūvio aptarti nebūtina.

3.4 Operacinis pjūvis

Operacinis pjūvis apibūdina, kaip sistema bus valdoma, administruojama ir palaikoma, kai ji veikia gamybinėje aplinkoje.

3.4.1 Operacinio pjūvio analizė

Pakeitimui bus reikalingas integracijų su trečiųjų šalių programomis palaikymas. Taip pat reikės konfigūruoti įvairias prieigas bei prisijungimus. Migruoti duomenų nereikės, nes sistema toliau veiks tokiu pačiu būdu, tik prisidės papildomos lentelės duomenų bazėje. Taip pat reikės įdiegti priežiūrai skirtas programas, kurios padės užtikrinti nenutrūkstamą mūsų programos ir kitų sąsajų veikimą.

4. Testavimas

Siekdami užtikrinti, kad mūsų sistema veiktų taip, kaip tikėtasi, ir pagerinti bendrą jos kokybę, įdiegėme automatinį testavimą įtraukdami UNIT ir integravimo testus. Automatiniai testai leidžia iš karto pastebėti bent kai kurias klaidas, kurių nepastebėjome kurdami ir taip sutaupyti laiko, kurį reiktų skirti rankiniam testavimui.

4.1 Automatiniai Testavimo Įrankiai

Įrenginiams ir integracijos testavimui įgyvendinti savo projekte panaudojome keletą išorinių paketų, nes norėjome patogesnio ir platesnio sprendimo nei siūlomas vanilla .NET. Norėdami pridėti testavimą ir vykdyti testus, naudojame paketą xUnit NuGet. Objektų integracijos testavimui naudojame MoQ biblioteką, o patogesniems testams naudojame paketą Fluent Assertions.

4.2 Automatizuotas reguliarus testavimas

Atlikdami ankstesnį darbą, jau buvome nustatę automatizuotą programinės įrangos darbo eigą, naudodami „Github Actions“, kad palaikytume kodo kokybę ir sukurtume sistemos kodo testus. Mes nustatėme, jog šie bandymai būtų automatiškai vykdomi kiekvienoje „push“ ir „pull“ užklausose, tačiau kodo stiliaus testus vykdome tik toms kodo dalims, kurios pasikeitė, kad sutrumpėtų testavimo laikas. Po to, kai parašėme testus naudodami anksčiau minėtus įrankius, mums tereikėjo pakoreguoti tai, jog galėtume automatiškai vykdyti kiekvienos versijos testus.

4.3 Implementuoti Testai

Kuriant testus, mūsų prioritetas buvo suprasti kurias sistemos dalis yra svarbiausia ištestuoti artimiausiu metu, nes neturėjome resursų idiegti pilną sistemos testavimą. Nustatėme, kad pirmenybė turėtų būti teikiama pakeitimams, kuriuos įgyvendinome šio darbo metu, o vėliau testavimas turėtų būti sukoncentruotas į mūsų anksčiau įdiegtas funkcijas ir galiausiai į pirmines sistemos funkcijas, kurios buvo sukurtos iki mūsų įsitraukimo į projektą. Šiuo metu mes įdiegėme apmokėjimo UNIT ir integravimo testus.

5. Sistemos architektūra

Kadangi šio projekto pirminė paskirtis buvo užduotis, kuri buvo sukurta neturint omenyje konkrečios programinės įrangos architektūros. Tačiau išanalizavę sistemą nustatėme, kad atitinka kai kuriuos gerai žinomus architektūros modelius. Tęsiant mūsų projektą, sukūrėme šį architektūros dokumentą, kad geriau suprastume sistemą ir sukurtume ją efektyvesnę ir nuoseklesnę.

5.1 Architektūriniai Stiliai

Nors kai kur architektūriniai stiliai ir architektūriniai modeliai yra tas pats, mūsų darbe šie dalykai išskiriami. Architektūriniai stiliai nusako kodo organizuotumą, o į architektūrinius modelius žiūrima kaip į būdą išspręsti pasikartojančias architektūrines problemas.

5.1.1 Klientas - serveris

Mūsų sistemos struktūra atitinka trijų pakopų architektūrą. Ši architektūra sudaryta iš trijų (reprezentacinio, loginio bei duomenų) lygmenų, kuriuos galima susieti su įprastais programinės įrangos elementais – naudotojo sąsajos (front end), posistemės (backend) bei duomenų bazės valdymu. Nors mūsų nerealizuota su serveriais ir veikia tik lokaliai, pasirinkta struktūra įgalina kūrėjus, padarius menkus pakeitimus, paleisti sistemą internete. Manome, jog ši struktūra yra patogi, kadangi kiekvieną pakopą vienu metu gali kurti atskira kūrimo komanda ir, jei reikia, ją atnaujinti arba keisti, nedarant įtakos kitoms pakopoms.

5.1.2 Orientuota į duomenų bazę

Sistemos architektūros bendros atminties aspektas buvo sukurtas orientuojantis į duomenų baze. Tai yra dažnas architektūrinio stiliaus pasirinkimas, nes daugelis žiniatinklio programų turi centrinę reliacinę duomenų bazę, kurioje saugomi visi sistemos duomenys. Mūsų sistema šiuo metu saugo duomenis lokaliai priglobtoje MSSQL duomenų bazėje. Manome, kad šis pasirinkimas yra tinkamas sistemai, nes mūsų sistema nėra sudėtinga ar didelė.

5.1.3 REST

Sistema iš dalies įgyvendina REST stilių. Kadangi internetinė aplikacija vadovaujasi REST principais (turi kliento-serverio architektūrą, yra be būsenos, nes serveryje nesaugoma jokios seanso būsenos, sistema taip pat naudoja vienodas sąsajas ir daugiasluoksnę sistemą – visa tai buvo pasiekta naudojant ASP.NET sistemą, kuri automatiškai leidžia nustatyti REST

programą). Manome, jog REST stilius atitinka mūsų sistemos poreikius, ypatingai paprastumo ir patikimumo privalumus.

5.1.4 Architektūrinis modelis

Tam, kad galėtume geriau suprasti sistemos architektūrą, su komanda išanalizavome, kokie sistemos vystymo šablonai buvo panaudoti. Tai padėtų implementuoti pakeitimą, kadangi naujai parašytas kodas nebūtų priešingas parašytajam, tenkintų funkcinius ir kokybės reikalavimus. Norėdami išanalizuoti architektūrinį modelį, išnagrinėjome pirminių programos kūrėjų kodą ir jame naudojamus principus.

5.1.5 MVC

Architektūrinis modelis labiausiai tinkantis mūsų sistemai yra MVC (model-view-controller) architektūros tipas. Šis modelis buvo naudojamas kuriant vartotojo sąsają. Šis modelis yra vienas populiariausių būdų kurti internetines aplikacijas, nes juo kurti sistemas yra paprasčiau, o tai ypač padeda pradedantiesiems programuotojams. Taip pat, sistemos logiką išskaidant į tris susijungiančias dalis, dalinius informacijos atvaizdavimas vartotojams tampa paprastesnis. Tai mums leido nesunkiai implementuoti apmokėjimo pakeitimą, nes visa informacija apie užsakymus ir vartotojus jau buvo saugoma sistemoje.

5.1.6 CRUD

Logiškai naudotojo sąsajai realizuoti ir atvaizduoti tinkamą informaciją buvo pasitelktas CRUD metodas. Šis architektūrinis modelis teigia, jog visą informacijos saugojimo ir gavimo funkcionalumą gali pasiekti keturios operacijos – sukurti, skaityti, pakeisti, ištrinti (create, read, update, delete). Tai palengvina sistemos dizainą, bei atitinka mūsų sistemos modelį, kadangi dauguma esybių reikalauja arba galėti siųsti informaciją į duomenų bazę (kurti POST užklausas), arba gauti informaciją iš duomenų bazės ir ją atvaizduoti vartotojui. Vienintelis šio modelio trūkumas yra tai, jog visa logika yra apibendrinta informacijos saugojimu arba atvaizdavimu vartotojui.

5.2 Architektūros perspektyvos

Nuodugnesnei sistemos analizei pasiekti, apžvelgėme skirtingas perspektyvas ir įvardijome problemas bei grėsmes. Šie veiksmai padės ateityje generuoti atitinkamus sprendimus.

5.2.1 Saugumas

Siekdami užtikrinti jautrių duomenų saugumą, patikimai valdyti ir atkurti sistemą nuo saugumo pažeidimo išanalizavome saugumo perspektyvą.

5.2.2 Asmeniniai vartotojo duomenys

FoodKept sistemai, kaip bet kokiai vartotojo duomenis renkančiai sistemai, būtina užtikrinti jų apsaugą. Labiausiai pažeidžiami duomenys FoodKept sistemoje yra elektroninis paštas bei slaptažodžiai. FoodKept sistemoje slaptažodžiai yra apsaugoti naudojant hašavimo funkciją, taip pat, naudojama Entity Framework Core sistema užtikrina apsaugą nuo SQL injekcijų.

5.2.3 Neautorizuota Prieiga

Kitas svarbus aspektas yra tai, kad vartotojai neturėtų leidimo prieiti prie duomenų, kurių jie neturėtų pasiekti. Kad taip nenutiktų, buvo įdiegti skirtingos vartotojo rolės. Tai buvo padaryta rolių valdymo sistema, kuri pateikiama kartu su ASP.NET Core sistema ir yra patikimesnė už nežinomas trečiųjų šalių sistemas.

Šiuo metu prisijungimo sistema neriboja bandymų prisijungti. Tai pašalina pažeidžiamumą, leidžianti mums naudoti „brute force“ metodiką bandant pasiekti kitų vartotojų paskyras. Norint išspręsti šią grėsmę, gali būt įdiegta paprasta funkcija, pvz., reikalaujama patvirtinimo el. paštu po tam tikro skaičiaus nesėkmingų prisijungimų.

5.2.4 Piktybinės atakos

Šiuo metu, sistema vis dar turi saugumo problemų: kol kas vis dar nėra jokios apsaugos nuo paslaugos trikdymo (DoS/DDoS), ar nuo kitų panašių atakų, kurių dėka būtų galima sutrikdyti mūsų programėlės prieinamumą. Paprasčiausia DoS ataka galėtų būti įvykdyta vieno asmens, kuris daug kartų per mažą laiko tarpą bandytu prisijungti prie sistemos su neteisingais duomenimis. Ar ši ataka pasiteisintu, neįmanoma pasakyti, tačiau kol kas apsaugos nuo tokių atakų nėra. Tai teoriškai galėtų dar labiau prailginti neplanuotą puslapio nepasiekiamumą (downtime).

5.2.5 Nepasiekiamumas

Neplanuotas puslapio nepasiekiamumas (downtime) yra problema, kadangi gali atsirasti ne viena netikėta sistemos klaida. Taip pat, sistemos taisymo, keitimo, ar naujų funkcijų

implementavimas turės būti atliktas, kai sistema išjungta, kadangi neturime būdų sistemos keisti tuo pat metu, kol puslapis yra įjungtas.

5.2.6 Greitaveika ir plėtra

Norėdami užtikrinti malonią vartotojų patirtį naudojant mūsų programėlę, išanalizavome sistemos plėtros ir veikimo perspektyvas.

5.2.6.1 Greitaveika

Sistemos greitaveika nėra pilnai ištobulinta. Gijos (threads) naudojamos tik keliose, greitaveikai nesvarbiose vietose. Nors daugelyje vietų gijų nereikia, vietose, kur kraunama informacija pereinant į skirtingus puslapius, gijos pagreitintų puslapio užsikrovimą, kadangi nereikėtų laukti visos informacijos, kuri turi būti pavaizduota. Žinoma, greitaveikos problemos atsirastų tik tuo atveju, jei web programėlę naudotų didelis naudotojų skaičius vienu metu, kadangi serverio apkrovimas prisidėtų prie informacijos laukimo laiko. Gijų nebūvimas sistemai teikia vieną privaluma – sistemos veikimas yra nuspėjamas ir lengviau palaikomas (iš progrramuotojų pusės).

5.2.6.2 Plėtros apribojimai

Daugelis sistemos funkcijų iš duombazės nuskaito visą informaciją (kurios prašoma), o po to ją atfiltruoja sistemos viduje. Tai nėra reali problema, kol skaitomas ar rašomas nedidelis infomacijos kiekis, tačiau tai stipriai apsunkina sistemos plėtimą, bei gali versti vartotojus ilgai laukti norimos informacijos, kol sistema ją užkraus. Tai galėtu būti išspręsta keičiant kodo dalis, kur informacija gaunama, jas pakeičiant SQL užklausomis, o ne visos informacijos gavimu.

5.2.7 Pakeitimas ir perspektyvos

Kadangi neatsirado jokios jautrios informacijos, kurią sužinojus būtų galima pakenkti mūsų aplikacijos naudotojams, naujas programos pakeitimas nedaro įtakos esamiem saugumo reikalavimam, sistemos naudojimas ir duomenų saugumo aspektai išlieka nepakitę.

6. Atsekamumo matrica

Norėdami patikrinti, ar išnagrinėjome visus pjūvius ir perspektyvas, sukūrėme pjūvių ir perspektyvų atsekamumo matricą, parodytą 1 lentelėje.

	Pjūvis	Konteksto	Funkcionalumo	Plėtojimo	Informacijos	Diegimo	Operacinis
Perspektyva							
Saugumo		**X**	**X**		**X**		
Veikimo ir plėtimosi		**X**	**X**	**X**	**X**		
Prieinamumo		**X**			**X**		
Evoliucijos (raidos)		**X**	**X**		**X**		

Lentelė 1

7. Rezultatai

Atlikdami šį laboratorinį darbą pasiekėme šiuos rezultatus:

1. Apibrėžėme sistemos architektūrą, pakeitimą ir nefunkcinius reikalavimus atsižvelgiant į architektūrinius stilius.
2. Pateikėme pjūvius bei perspektyvas bei argumentavome, nurodėme motyvus, skatinusius vaizduoti pasirinktą architektūrą.
3. Implementuotavome pakeitimą
4. Implementavome automatizuotą integravimą ir sistemos testus, naudodami CI/CD procesus.

8. Išvados

Iš šio laboratorinio darbo rezultatų galime padaryti tokias išvadas:

1. Dabartinė sistema jau atitinka architektūros stilius ir modelius, kurie yra tinkami jos plėtrai, tačiau patobulinimai ir nauji architektūriniai sprendimai gali būti pritaikyti naudotojų skaičiui išaugus.
2. Pakeitimas tinka esamai architektūrai ir nėra priešiškas su joje vyraujančiais modeliais.