Міністерство освіти і науки України Сумський державний університет Навчально-науковий інститут бізнес-технологій «УАБС» Кафедра економічної кібернетики

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему «Автоматизація процесу підвищення безпеки проведення фінансових операцій в мережі Internet»

Виконав студент 2 курсу, групи ЕК.м-61а
(номер курсу) (шифр групи)

Спеціальності 051«Економіка («Економічна кібернетика»)»

Чещевий Євген Ігорович
(прізвище, ініціали студента)

Керівник к.е.н., доцент, Бойко А.О.
(посада, науковий ступінь, прізвище, ініціали)

Суми – 2018

ЗМІСТ

ВСТУП

З кожним роком здійснення фінансових операцій через мережу Інтернет набирають все більше популярності. Разом з цим збільшилась й кількість шахраїв. Одна з ключових проблем у сфері електронної комерції – це продаж неіснуючих товарів. За статистикою, кожен шостий українець був обманутий при здійсненні фінансових операцій в Інтернеті. Проблему шахрайських дій в мережі Інтернет досліджували такі вчені, як Мусієнко О. Л., Паламарчук Л. П., Постіл Н. С., Романенко Т.В. Самойлов С. В., Том Зеллер Мл., Джеймі Довард, Кейлі Бурк та інші [8, 9, 10, 11, 12, 13, 14, 15].

Об’єктом дослідження виступає процес здійснення фінансових операцій в мережі Інтернет. Предметом дослідження є інформаційні системи та технології здійснення фінансових операцій в інтернеті.

Мета даної дипломної роботи полягає в розробці автоматизованого рішення проблеми безпеки здійснення фінансових операцій в мережі Інтернет.

Відповідно до поставленої мети основні завдання сформульовані наступним чином:

розглянути, що собою являють фінансові операції;

визначити існуючі способи здійснення фінансових операцій в мережі Інтернет;

дослідити механізми забезпечення безпеки здійснення фінансових операцій в мережі Інтернет;

сформулювати модель та вибрати архітектури вирішення задачі підвищення безпеки;

ідентифікувати програмне та технічне забезпечення для розробки прототипу;

реалізувати прототип для вирішення проблеми здійснення фінансових операцій;

навести рекомендації щодо практичного застосування розробленого прототипу;

Джерелами інформації, які були використані при написанні дипломної роботи є праці вітчизняних та зарубіжних вчених, нормативно-правові документи, тези доповідей, наукові публікації в періодичних виданнях, ресурси Інтернет.

1 ДОСЛІДЖЕННЯ СТАНУ ОБ’ЄКТА АВТОМАТИЗАЦІЇ ТА ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ АВТОМАТИЗАЦІЇ

1.1 Теоретичні основи та сутність фінансових операцій

Фінансова операція – це угода та інші дії громадян або юридичних осіб з фінансовими засобами незалежно від форми і способу їх здійснення, пов'язані з переходом права власності та інших прав, включаючи операції, пов'язані з використанням фінансових коштів в якості засобу платежу, в тому числі:

здійснення міжнародних грошових переказів на територію держави і з території держави;

отримання і надання фінансових кредитів;

переведення неторгового характеру з території держави і на територію держави, а також в межах держави, включаючи суми заробітної плати, пенсії, аліментів, спадщини, а також інші аналогічні операції;

придбання цінних паперів [33].

Трансакції (фінансові операції) обумовлюються грошовими платежами (трансфери, розрахунки та ін.) та рухом капітальних ресурсів. Об’єктами виступають різноманітні фінансові активи, до яких належать іноземна валюта, національні гроші, цінні папери, нерухомість та дорогоцінні метали.

Фінансові операції поділяють на такі види: операції з переказу грошей, інвестиційні операції, спекулятивні операції та операції за капіталом [42].

Операції з переказу грошей – це усі види та форми розрахунків (операції з обміну "гроші – товар") та трансфери (рух грошей в одному напрямі) [43].

Інвестиційні фінансові операції пов'язані з переміщенням капіталу з метою його приросту. Це середньострокові та довгострокові вкладення капіталу. До яких належать лізинг, кредит, оренда, франчайзинг, траст, рента та всі операції які тривають понад 180 днів.

Спекулятивні операції – це короткострокові фінансові дії з отримання прибутку у вигляді різниці у процентах з отриманих кредитів. До них належать валютний арбітраж, процентний арбітраж, операції своп, валютна спекуляція тощо [44].

Операції з капіталом охоплюють капітальні трансферти та обмін нефінансовими активами. Ці операції спрямовані на управління капіталом в умовах ризику та невизначеності економічної кон'юнктури [45].

При проведенні фінансових операцій треба також враховувати, що в багатьох країнах встановлюються пільгові умови в офшорних зонах. Річ у тім, що при проведенні жорсткої монетарної, фіскальної та кредитної політики приватний капітал намагається знайти сприятливіші умови інвестування в інших державах. Певну роль в цьому напрямі відіграє і зростання тіньового капіталу, обсяги якого в Україні оцінюються в межах 30% від офіційного товарообігу. З метою залучення капіталу шляхом створення для нього більш сприятливих режимів і створюються офшорні зони, перші з яких почали діяти в другій половині XX ст. А в останні роки їх кількість помітно зросла.

Офшорна зона (юрисдикція) – це країна або окрема територія країни, де для певних типів іноземних компаній встановлено пільги з оподаткування, зменшені або зовсім відсутні вимоги до бухгалтерського обліку і аудиту, частково або повністю відмінені митні і торговельні обмеження [46].

Органи державного управління, які створюють такі зони, намагаються залучити додатковий капітал на свою територію з метою підвищити рівень зайнятості, одержати доходи, підвищити попит на товари та послуги та ін.

Досягається це за рахунок скорочення оподаткування, зменшення вимог до заснування фірми, скорочення юридичних та фінансових обмежень їх діяльності. Залучення капіталу за таких умов одержало розвиток в таких державах, як Кіпр, Мальта, Мен, Коста-Рика, Багамські, Віргінські та Бермудські острови та ін. Далі до них приєдналися держави та окремі території у складі держав: Ірландія, Бахрейн, Андорра, Ліхтенштейн, Ліберія, Сінгапур, Інгушетія та ін.

Офшорна компанія – це компанія, зареєстрована в юрисдикції з низьким рівнем оподаткування за умов здійснення фіксованої оплати зборів у період заснування. Умовою надання пільг для компанії є те, що її власниками мають бути іноземні особи, а прибутки створюються за межами зони [47].

Найчастіше офшорний бізнес одержує розвиток в банківській справі, страхуванні, торгівлі, будівництві, наданні транспортних послуг, інвестиційній діяльності, проведенні операцій з цінними паперами та ін.

Основними організаційними формами підприємств в офшорних зонах є офшорна компанія з обмеженою відповідальністю, офшорний філіал і офшорне партнерство.

Важливим для іноземного інвестора є те, що в зоні забезпечується конфіденційність діяльності власників компанії. Під час її заснування акціонерами часто записують не справжніх інвесторів, а юристів, які и реєструють, і трастові компанії. Відповідальними особами компанії стають номінальні директори, які не здійснюють контроль і управління, а виконують накази реальних власників. Обмеження в одержанні інформації про справжніх власників офшорної компанії практично не дає можливостей (за виключенням проведення спеціального розслідування, узгодженого з міжнародними судовими установами) визначити коло пов'язаних осіб.

Офшорних компанії з погляду підприємницької діяльності мають такі переваги:

анонімність і конфіденційність під час проведення фінансових операцій;

відсутність оподаткування або наявність його за низькими ставками;

послаблення контролю за проведенням валютних операцій; одержання прибутків у твердій валюті та ін.

За принципами оподаткування офшорні зони поділяються на три групи:

з фіксованою платою і відсутністю оподаткування;

з наявністю оподаткування за низькими ставками;

змішані системи, які охоплюють як фіксовані виплати, так і оподаткування доходів офшорних компаній.

Деякі фінансові операції можуть підлягати фінансовому моніторингу.

Об’єктом фінансового моніторингу виступають підозрілі фінансові операції [48]. У світовій практиці операція вважається підозрілою за одночасної наявності двох ознак: кількісної (обсяг здійснюваної угоди) та якісної.

Закон України «Про запобігання та протидію легалізації (відмиванню) доходів, одержаних злочинним шляхом, фінансуванню тероризму та фінансуванню розповсюдження зброї масового знищення» дає специфічне визначення терміну «фінансова операція», під якою розуміють будь-яку операцію, пов’язану зі здійсненням або забезпеченням здійснення платежу за допомогою суб’єкта первинного фінансового моніторингу, зокрема:

- внесення або зняття депозиту (внеску, вкладу);
- переказ грошей з рахунка на рахунок;
- обмін валюти;
- надання послуг з випуску, купівлі або продажу цінних паперів та інших видів фінансових активів;
- надання або отримання позики або кредиту;
- страхування (перестрахування);
- надання фінансових гарантій та зобов’язань;
- довірче управління портфелем цінних паперів;
- фінансовий лізинг;
- здійснення випуску, обігу, погашення (поширення) державної та іншої грошової лотереї;
- надання послуг з випуску, купівлі, продажу й обслуговування чеків, векселів, платіжних карток, грошових поштових переказів та інших платіжних інструментів;
- відкриття рахунка [15].

В Україні запроваджено двосторонній механізм вивчення та звітування про підозрілі фінансові операції [49]. Цей механізм полягає в обов'язковому та внутрішньому фінансовому моніторингу. На суб'єктів первинного фінансового моніторингу покладено обов'язок звітувати на власний розсуд про операції, що підлягають обов'язковому фінансовому моніторингу, а також вимога звітувати про фінансові операції, що стали об'єктом внутрішнього фінансового моніторингу.

Обов'язковий фінансовий моніторинг — це сукупність заходів спеціально уповноваженого органу (ДКФМ) з питань фінансового моніторингу з аналізу інформації щодо фінансових операцій, яка надається суб'єктами первинного фінансового моніторингу, а також заходів з перевірки такої інформації відповідно до законодавства України [48].

Внутрішній фінансовий моніторинг — це діяльність суб'єктів первинного фінансового моніторингу з виявлення фінансових операцій, що підлягають обов'язковому фінансовому моніторингу, та інших фінансових операцій, що можуть бути пов'язані з легалізацією (відмиванням) доходів.

Ознаки операцій, що є об'єктом обов'язкового фінансового моніторингу, визначено у вітчизняному законодавстві у вигляді чітких кількісних та формальних визначень операцій [48].

По-перше, фінансова операція підлягає обов'язковому фінансовому моніторингу, якщо вона підпадає під такі кількісні ознаки:

сума, на яку вона проводиться, дорівнює чи перевищує 80000 гривень;

дорівнює чи перевищує суму в іноземній валюті, еквівалентну 80 000 гривень.

По-друге, фінансова операція підлягає обов'язковому фінансовому моніторингу, якщо має одну чи більше формальну ознаку, що визначена законодавством [15].

Ці ознаки розробляються державою в особі уповноваженого органу ДКФМ та закріплені в Законі України «Про запобігання та протидію легалізації (відмиванню) доходів, одержаних злочинним шляхом,

фінансуванню тероризму та фінансуванню розповсюдження зброї масового знищення». До них, відповідно до статті 11 зазначеного Закону, належать:

переказ грошових коштів на анонімний (номерний) рахунок за кордон і надходження грошових коштів з анонімного (номерного) рахунка за кордону, а також переказ коштів на рахунок, відкритий у фінансовій установі в країні, що віднесена Кабінетом Міністрів України до переліку офшорних зон;

купівля (продаж) чеків, дорожніх чеків або інших подібних платіжних засобів за готівку;

зарахування або переказ грошових коштів, надання або отримання кредиту (позики), проведення фінансових операцій із цінними паперами у випадку, коли принаймні одна зі сторін є фізичною або юридичною особою, що має відповідну реєстрацію, місце проживання чи розташована в країні (на території), яка не бере участі в міжнародній співпраці у сфері запобігання та протидії легалізації (відмиванню) доходів, отриманих злочинним шляхом, та фінансуванню тероризму, або однією зі сторін є особа, що має рахунок у банку, зареєстрованому в зазначеній вище країні (на зазначеній вище території);

переказ коштів у готівковій формі за кордон з вимогою видати отримувачу кошти готівкою;

зарахування на рахунок коштів у готівковій формі із подальшим переказом їх того самого або наступного операційного дня іншій особі;

зарахування грошових коштів на рахунок чи списання грошових коштів з рахунка юридичної особи, період діяльності якої не перевищує трьох місяців від дня її реєстрації, або зарахування грошових коштів на рахунок чи списання грошових коштів з рахунка юридичної особи у випадку, якщо операції на зазначеному рахунку не проводилися з моменту його відкриття;

відкриття рахунка з внесенням на нього коштів на користь третьої особи;

переказ особою, за відсутності зовнішньоекономічного контракту, коштів за кордон;

обмін банкнот, особливо іноземної валюти, на банкноти іншого номіналу;

проведення фінансових операцій із цінними паперами

на пред'явника, не розміщеними в депозитаріях;

придбання особою цінних паперів за готівку;

виплата фізичній особі страхового відшкодування або отримання страхової премії; виплата особі виграшу в лотерею, казино або в іншому гральному закладі;

розміщення дорогоцінних металів, коштовного каміння та інших цінностей у ломбард.

Зауважимо, що обов'язкове звітування за власним бажанням поширюється лише на чітко визначений перелік типів операцій.

Для визначення операцій, що є об'єктом обов'язкового фінансового моніторингу, застосовано як підхід із визначенням кількісних та форматизованих ознак, так само об'єктом обов'язкового моніторингу є фінансові операції, що підлягають внутрішньому фінансовому моніторингу.

1.2 Особливості проведення фінансових операцій в мережі Інтернет

За останні кілька років оплата товарів і послуг через Інтернет набуває дедалі більшої популярності, оскільки такий спосіб доступний, зручний і економить час. Тому з'являються різні платіжні системи, які допомагають здійснювати фінансові операції.

Найпопулярнішими в українському сегменті мережі Інтернет (про це свідчить кількість реєстрацій) є такі системи електронних грошей: Webmoney, Яндекс.деньги, RBK Money, Ukrmoney, E-Gold і Paypal [50, 51, 52, 53].

Платіжна система в Інтернет (e-payment system) - це система здійснення розрахунків і платежів між комерційними структурами, фінансовими установами та користувачами мережі в процесі купівлі - продажу товарів та послуг через Інтернет [54].

Саме наявність платіжної системи дозволяє створювати повнофункціональні віртуальні торгові підприємства, в яких здійснюється весь технологічний процес купівлі-продажу товару чи послуги [55].

Платежі в мережі Інтернет повинні здійснюватися за дотримання ряду умов:

конфіденційність – під час проведення платежів в Інтернет дані покупця (анкетні та адресні дані, номер кредитної картки тощо) відомі тільки установам, які мають на це законне право;

цілісність – інформацію про купівлю ніхто не в змозі змінити;

збереження таємниці – повинен бути забезпечений захист повідомлень від несанкціонованого перегляду;

автентифікація – і продавці, і покупці повинні мати гарантію, що всі сторони, які беруть участь в угоді, є дійсно тими за кого себе видають;

авторизація – під час проведення платежу обов'язкове здійснення цього процесу, під час якого вимога на проведення трансакції (банківської операції) підтверджується або відхиляється платіжною системою. Авторизація дозволяє визначити наявність коштів у покупця;

багатоваріантність засобів оплати – покупець може оплатити придбання товару або послуги будь-якими доступними йому платіжними засобами;

гарантії ризиків продавця – продавець в Інтернет зазнає багатьох ризиків, пов'язаних, в основному, з несумлінністю покупця та відмовою його від товару;

мінімізація плати за транзакцію – плата за обробку трансакції замовлення і оплати товарів входить у вартість товару, тому зниження ціни трансакції збільшує конкурентоспроможність продавців.[1]

Для фінансових операцій в мережі Інтернет, крім традиційних банківських пластикових карток, використовують різноманітні системи електронних грошей.

Під електронними грішми мають на увазі еквівалент традиційних фіатних (паперових) грошових коштів, що перебувають в обігу всередині конкретної електронної платіжної системи (далі − ЕПС).

Сутність електронних грошей полягає в зберіганні грошової вартості на електронних носіях − смарт-картах або жорсткому диску комп'ютера. Їх обіг відбувається за допомогою комп'ютерних мереж, Інтернету, платіжних карт, електронних гаманців і пристроїв, що працюють з платіжними картами (банкоматів, POS-терміналів) [56]. По суті електронні гроші не є грошима в традиційному розумінні цього слова.

Справжні ж гроші залишаються в оператора ЕПС, тобто в разі розрахунків через ЕПС руху реальних грошей не відбувається, електронні гроші обмінюють на реальні після закінчення угоди через банки-партнери оператора.

Обіг електронних коштів може здійснюватися як за правилами, встановленими або узгодженими з державними центробанками, так і за власними правилами недержавних електронних платіжних систем.

Виокремлюють два види електронних грошей − державні та приватні. Державні виражені в національній валюті і є складовою державної платіжної системи, а приватні − це електронні одиниці недержавних платіжних систем, які своїми правилами регламентують їх емісію та обіг. Держава ніяк не забезпечує надійність або ліквідність приватних грошей [57].

Не визнають електронними грішми дисконтні картки торгівців, а також подарункові карти, картки автозаправних станцій (паливні картки), квитки для проїзду в міському транспорті, телефонні картки тощо, що приймають як засіб платежу виключно їх емітентами.

Вимоги, що регулюють випуск та використання електронних грошей в Україні, викладено у ст. 15 Закону України «Про платіжні системи та переказ

коштів в Україні» від 05.04.2001 р. № 2346-III (далі – Закон № 2346)

[35] та Положенні про електронні гроші в Україні, затвердженому постановою Правління Національного банку України від 04.11.2010 р. № 481 (далі − Положення № 481) [36]. Положення регулює діяльність, пов’язану з випуском електронних грошей в Україні, та запроваджує моніторинг за такою діяльністю.

Випуск електронних грошей в Україні мають право здійснювати лише банки. Банк надає своїм клієнтам такі фінансові послуги у сфері використання електронних грошей:

розповсюдження електронних грошей;

здійснення обмінних операцій з електронними грішми;

надання засобів поповнення електронними грошима електронних пристроїв;

приймання електронних грошей в обмін на готівкові/безготівкові кошти.

Згідно з вимогами п. 15.2 ст. 15 Закону № 2346 банк, який планує здійснювати випуск електронних грошей, зобов’язаний спершу узгодити з Національним банком України (далі – Нацбанк, НБУ) правила їх використання в порядку, встановленому нормативно-правовим актом НБУ. Відповідно до вимог Нацбанку суб’єкти, що здійснюють операції з електронними грішми (емітент електронних грошей, оператор, агенти, торгівці і користувачі), можуть робити це тільки за правилами, погодженими з НБУ [35].

Принцип роботи електронних грошей загалом простий. Будь-яке підприємство чи компанія може бути як у ролі торгівця, отримуючи електронні гроші за товари та послуги, так і в ролі користувача, оплачуючи ними свої покупки. Для цього компанія як юридична особа на підставі

договору, укладеного з банком (емітентом або агентом з розрахунків), відкриває два електронні рахунки (гаманці) − один для отримання електронних грошей як торгівця, а інший – для розрахунку як користувача.

Щоб здійснити платіж, необхідно спочатку придбати за реальні гроші платіжні засоби системи (зобов'язання). Вони зараховуються на електронний гаманець, а потім їх використовує користувач. Поповнювати електронний гаманець можна через відділення банків, з якими співпрацює система, з платіжної картки, з поточного банківського рахунку або через платіжні термінали [58].

У разі надходження електронних грошей на електронний гаманець їх можна перевести в готівку або на банківський рахунок.

Відповідно до п. 3.3 Положення № 481 користувачі − фізичні особи мають право використовувати електронні кошти для оплати товарів, а також переказувати їх іншим користувачам. Вони можуть переказувати електронні гроші іншим користувачам − фізособам з використанням наперед оплачених карток у сумі до 500 грн на день та не більше 4 тис. грн протягом одного місяця [36].

Користувачі − суб'єкти господарювання – юридичні особи мають право отримувати електронні гроші виключно в обмін на безготівкові кошти. Вони можуть використовувати такі гроші лише для здійснення оплати товарів.

Торгівці вправі приймати від користувачів як засіб платежу за товари електронні гроші, виражені лише у гривнях. Використовувати отримані кошти можна виключно для обміну на безготівкові кошти або повертати електронні гроші користувачам у разі повернення ними відповідно до Закону України «Про захист прав споживачів» товарів, придбаних у такий спосіб.

Перевести електронні гроші у гривні можна тільки за умови виведення з платіжної системи [2].

Також разом з електронними грошима зараз дуже швидко розвивається крипто валюти.

Криптовалюти – новий вид електронних грошей, що швидко набирає популярності [59]. Незважаючи на відсутність офіційної легалізації в Україні,

станом на початок 2017 р. країна визнана одним зі світових лідерів із застосування криптовалют [60]. За даними «CoinMarketCap» сьогодні в світі

існує 699 видів криптовалют, їх сумарна ринкова капіталізація збільшилася на 53% з початку 2017 р., з 17,7 до 27,144 млрд. дол. США при тому, що в січні 2016 р. становила 6 млрд. дол. США [61].

Серед найбільш популярних криптовалют у світі лідерами росту стали Bitcoin, Ethereum, DASH, Monero и NEM – їм належить 91% сумарної капіталізації. Ціна першої криптовалюти – біткоїна – «досягла абсолютного історичного максимуму і вже наближається до позначки 20000 дол. за 1 BTC (що перевищує ціну тройської унції золота)» [62].

Криптовалюту в Україні можна заробити, купити або подарувати. Раніше заробляти її було легше, у 2017 р. ситуація змінилася. В Україні «проблематично розплатитися криптовалютою за товари чи послуги, лише одиниці онлайн-магазинів і підприємств зазначають на своїх сайтах, що приймають криптовалюту як оплату товарів чи послуг». Власнику бізнесу можна використовувати оплату в новому виді валют, як це роблять багато західних магазинів.

У деяких країнах роботодавці вже виплачують своїм працівникам зарплати в криптовалюті, зазвичай тим, хто працює через інтернет – в Україні така практика лише починає поширюватися. Заробітну плату в криптогрошах серед українців сьогодні отримують здебільшого фахівці IT-сфери, які працюють на закордонні компанії [63].

Інколи криптовалюту можна заробляти й безоплатно: це невеликі подарункові суми, які залучають людей по всьому світі. Для цього досить лише ввести капчу або переглянути рекламу, і за це будуть нараховані, до прикладу, Сатоши (один стомільйонний біткоїна). Але це не є нормальним способом заробітку, оскільки у разі блокування сайту криптовалюта зникає. Подарунки можливі від великих сервісів, наприклад, за гарний пост або статтю, картинку, яка користується попитом у інших користувачів.

Купувати криптовалюту в Україні можна на різних біржах, в обмінниках. Обмінники не завжди вигідні, оскільки заробляють на різниці вартості, оптимальними є варіанти з біржами або купівля безпосередньо у

продавця з гарною репутацією. Варто пам’ятати про безпеку угоди; багато практиків рекомендують купувати криптовалюту у продавців у своєму місті [64].

Процес утворення нового платіжного інструменту, яким є криптовалюти, «задіює комп’ютерні потужності мільйонів учасників, заснований на даних відкритих електронних гаманців», що об’єднані в одну пірингову мережу за відсутності центрального сервера. Таким чином, вся робота з обліку, зберігання історії транзакцій розподіляється між усіма учасниками.

Одним із найважливіших аспектів функціонування криптовалюти є потужні сучасні криптографічні методи захисту, що забезпечують ідентифікацію власників та фіксацію факту їх зміни. Віра в неможливість зламу цього захисту нам видається дещо ідеалістичною без належної постійної IT-підтримки фахівців.

На початок 2017 р. найбільш поширеними в Україні були такі криптовалюти:

Bitcoin (BTC) – найперша зі створених криптовалют, упевнено утримує пальму першості у світі й на українському ринку. Ціна монети наближається до 15000 дол. США [37].

Ethereum (ETH) – криптовалюта, що була випущена лише в 2015 р. на базі технології Ethereum, яка належить до блокчейн-систем, заснованих на використанні розподілених баз даних. Ethereum може застосовуватися в різних сферах, при цьому також є валютою, альтернативною традиційним інструментам валютного ринку. Ціна – близько 32 дол. США. Зростанню інтересу до цієї криптовалюти сприяло надання можливості торгувати парою ETH / USD найбільшою у світі соціальною торгово-інвестиційною мережею «eToro», а також блокчейн об’єднання «Enterprise Ethereum Alliance», куди увійшли Microsoft, Intel і Accenture [38].

Рекордно швидке зростання криптовалюти Dash аналітики пов’язують із виходом оновлення Dash v12.1 – «Sentinel», покликаного підвищити швидкість

роботи, поліпшити рівні забезпечення секретності і координації в мережі. Ціна Dash сьогодні наблизилася до 100 дол. США [39].

Monero (XMR) – криптовалюта з відкритим вихідним кодом, що відрізняється безпекою й анонімністю. Стабільно входить до п'ятірки монет із високою ринковою капіталізацією. Ціна – 18 дол. США [40].

Ripple (XRP) – платіжна система з відкритим вихідним кодом, де внутрішньою платіжною одиницею є однойменна криптовалюта.

Технологія Ripple ґрунтується на «принципі довіри». Ідея полягає в можливості обміняти свою валюту на будь-яку іншу за найвигіднішим курсом в один клік, додавши потрібний шлюз. Ціна – 0,006440 дол. США [41].

Перевагами криптовалют є відсутність зовнішнього чи внутрішнього адміністратора, анонімність, можливість для інвестування, відносна надійність, неопосередкованість, незалежність від політичної ситуації, оперативність, спрощення транскордонних платежів та багато інших.[4]

1.3 Механізми забезпечення безпеки в мережі Інтернет

Інформаційна безпека означає можливість протистояти спробам нанесення збитків власникам або користувачам системи при різних навмисних або ненавмисних впливах на неї.

Система захисту інформації повинна забезпечувати безперервний захист інформації щодо переказу грошей на усіх етапах її формування, обробки, передачі та зберігання. Електронні документи, що містять інформацію, яка відноситься до банківської таємниці або є конфіденційною, повинні бути зашифрованими під час передавання їх за допомогою телекомунікаційних каналів зв'язку. Порядок захисту та використання засобів захисту інформації щодо переказу визначається Законами України "Про платіжні системи та переказ грошей в Україні" [5].

Захист інформації забезпечується суб’єктами переказу грошей шляхом обов’язкового впровадження та використання відповідної системи захисту. Система захисту інформації складається з:

законодавчих актів України та інших нормативно-правових актів, а також внутрішніх нормативних актів суб’єктів переказу, що регулюють порядок доступу та роботи з відповідною інформацією, а також відповідальність за порушення цих правил;

заходів охорони приміщень, технічного обладнання відповідної платіжної системи та персоналу суб’єкта переказу;

технологічних та програмно-апаратних засобів криптографічного захисту інформації, що обробляється в платіжній системі [66].

Система захисту інформації повинна забезпечувати:

цілісність інформації, що передається в платіжній системі, та компонентів платіжної системи;

конфіденційність інформації під час її обробки, передавання та зберігання в платіжній системі;

неможливість відмови ініціатора від факту передавання та отримувачем від факту прийняття документа на переказ, документа за операціями із застосуванням засобів ідентифікації, документа на відкликання; забезпечення постійного та безперешкодного доступу до компонентів платіжної системи особам, які мають на це право або повноваження, визначені законодавством України, а також встановлені договором [67].

Розробка заходів охорони, технологічних та програмно-апаратних засобів криптографічного захисту здійснюється платіжною організацією відповідної платіжної системи або іншою установою на її замовлення відповідно до законодавства України та вимог, встановлених Національним банком України [68].

Слабким місцем мережевої інфраструктури є незахищені канали зв'язку, де існує можливість появи неіснуючих платежів, перехоплення та

несанкціонованого використання платіжних даних, інших проявів шахрайства та зловживань. Такі випадки траплялися в Internet.

Існують два шляхи захисту передачі інформації: ізолювання мережі, яка використовується для обробки платежів, тобто використання приватних мереж, і шифрування даних [69].

Загальновизнано, що асиметричні алгоритми шифрування забезпечують вищий рівень надійності та безпеки. Найбільш поширене шифрування за допомогою відкритих ключів.

Для розшифрування використовуються таємні ключі. Кількість можливих комбінацій, придатних для розшифрування зашифрованої за допомогою відкритих ключів інформації, залежить від довжини ключа в бітах. Вживання довших послідовностей у ролі ключів дозволяє випереджати можливості сучасних комп'ютерів, забезпечуючи надійний захист найважливішої інформації [69].

Донедавна в Internet для захисту інформації найчастіше застосовувалися ключі довжиною 40 біт. Але ці ключі були легко розкриті (слід зазначити, що використання такого ненадійного засобу було наслідком дуже жорстких обмежень на експорт зі США потужних шифрувальних засобів). Нині в Internet розповсюджені ключі довжиною 1024 біт або навіть довші, що робить практично неможливим їх розкриття [71].

У платіжних системах важливе завдання - ідентифікація учасників платежу (автентифікація). Це робиться для обмеження доступу до платіжних засобів, надаючи право на такий доступ лише їх власникам, та для забезпечення можливості одержувати повідомлення про платіжні операції лише тим, кому вони адресовані.

Надійна автентифікація сприяє також підвищенню рівня взаємної довіри між учасниками електронної комерції. На поточний момент в Internet застосовуються різноманітні засоби автентифікації. До них належить персональний ідентифікаційний код (PIN-код), який потрібно

ввести перед виконанням фінансової операції. Але незначна довжина таких кодів дозволяє шахраям досить легко долати такі методи захисту.

Дуже ефективним засобом захисту є електронний підпис. Його застосування передбачає наявність у того, хто цей підпис перевіряє, відкритого ключа, що відповідає застосованому особистому ключу (підпису) відправника електронного документа. Однак, при отриманні такого відкритого ключа не виключено шахрайство, внаслідок якого буде передано підроблений відкритий ключ, який автентифікуватиме підроблені повідомлення.

Щоб запобігти цьому, застосовується практика сертифікування (підтвердження) відкритих ключів з боку уповноваженої на це і заслуговуючої довіри установи або організації.

Таку установу чи організацію називають сертифікуючою, і вона ставить свій електронний підпис на відкритих ключах, які надаються користувачам платіжної системи. Відкритий ключ сертифікуючої установи повинен надаватися з використанням надійно захищених каналів. Ця установа сертифікує відкриті ключі учасників системи лише після перевірки їх тотожності з використанням захищених каналів або інших методів, що забезпечують достатній рівень безпеки, і лише підтверджені нею відкриті ключі вважаються чинними.

За принципами використання криптографічний захист може бути вбудованим у платіжну систему або бути додатковим механізмом, який може відключатися.

Виділяють дві групи криптографічних алгоритмів:

Загальні криптоалгоритми мають повністю відкритий алгоритм, а їх криптостійкість визначається ключами шифрування.

Спеціальні криптоалгоритми мають таємний алгоритм шифрування.

Загальні криптоалгоритми розподіляються на дві групи:

Симетричні алгоритми – криптографічні алгоритми, для яких шифрування і розшифрування виконуються одним ключем. Відправник і отримувач повідомлення повинні мати один і той самий ключ [72].

Асиметричні алгоритми – криптографічні алгоритми, для яких шифрування і розшифрування виконуються за допомогою різних ключів [73].

Криптографічні алгоритми використовуються з метою:

по-перше, шифрування інформації (приховування змісту повідомлень і даних);

по-друге, забезпечення захисту даних і повідомлень (інформації) від модифікації, викривлення або підробки. Для асиметричних криптографічних алгоритмів є можливість сформувати додаткову інформацію у вигляді електронного цифрового підпису.

Виділяють два методи розподілу криптографічних ключів між учасниками платіжної системи.

Метод базових-сеансових ключів використовується для розподілу ключів симетричних алгоритмів шифрування. Для розподілу ключів вводиться ієрархія ключів: ключ шифрування ключів і ключ шифрування даних [74].

Метод відкритих ключів використовується для розподілу ключів як для симетричного, так і для асиметричного шифрування. Використання методу дає можливість кожне повідомлення шифрувати окремим ключем симетричного алгоритму та передавати цей ключ із самим повідомленням у зашифрованому асиметричним алгоритмом вигляді. [5]

2 ПРОЕКТУВАННЯ СИСТЕМИ АВТОМАТИЗАЦІЇ ЕКОНОМІЧНОГО ОБ'ЄКТА

2.1 Модель процесу підвищення безпеки проведення фінансових операцій в мережи Інтернет

Здебільшого науковці акцентують свою увагу на питаннях вивчення слідів, які залишаються при вчинені шахрайства в мережі Інтернет. В тому числі на «Інтернет-аукціонах». Вони розкривають зміст протиправних дій, виділяють та вивчають основні дії які вчиняються при скоєні злочину.

Проаналізувавши деякі дослідження, можна сказати, що не було запропоновано, як можна вирішити, або хоча б зменшити кількість випадків шахрайства на просторах Інтернету.

Як правило, для вчинення шахрайських дій, зловмисники використовують відомі «Інтернет-аукціони» та дошки об'яв. Такий підхід одночасно дозволяє розташувати до себе покупців, використовуючи авторитетність аукціону, а також охопити велике коло потенційних жертв.

Розповімо детальніше про декілька з видів шахрайств. Шахрайство з авансовим платежем/передплатою – вас просять надіслати кошти, щоб внести авансовий платіж за надання послуги або отримання товару. Шахраї видають себе за представників фальшивих кредитних компаній. Вони вимагають перерахувати «аванс» перед наданням кредиту. Споживачі платять, але такі кредити ніколи не надаються.

Нігерійські листи щастя – ви одержуєте лист від адвоката. Найчастіше вони написані поганою англійською від імені якогось «адвоката» і містять зворушливу історію про те, як у Нігерії ваш нібито далекий родич трагічно загинув, залишивши вам у спадок величезне майно. Але спочатку вам треба надіслати гроші на обслугоування рахунку [75].

Онлайн-продажі – шахрай не висилає гроші за товар. Прикидаючись покупцем, пише продавцю, що готовий вислати вам поштою чек, який покриває і вартість покупки і суму, яка необхідна для пересилання. Продавець і справді отримує чек, оплачує відправку товару, а потім з'ясовується, що чек був фальшивий. І продавець втрачає як товар, так і гроші.

Фішинг – шахраї надсилають потенційній жертві лист з банку або платіжної системи, наприклад, PayPal. У листі написано, що з рахунком жертви є якісь проблеми, для вирішення яких необхідно пройти по посиланню з листа. Найчастіше ця вимога супроводжується загрозою блокування банківського рахунку.

Посилання, як правило, веде на фальшивий сайт банку — як правило, він виглядає в точності, як справжній сайт, та і адреса відрізняється лише однією літерою. Довірлива жертва вводить справжні логін та пароль, тим самим повідомляючи їх шахраям [76].

Для більш повного розуміння суті шахрайських дій, надаємо узагальнену характеристику принципів діяльності таких сайтів і мобільних додатків. Ці сайти надають можливість своїм користувачам виставляти лоти на продаж та вести торги за вже виставлені лоти.

Для цього всі користувачі повинні:

1) зареєструватися на сайті (процес реєстрації для кожного окремого інтернет порталу встановлюється адміністрацією порталу.

2) підтвердити реєстрацію;

3) виставити лот, встановивши його початкову вартість (також можливі випадки «фіксованої вартості» – вартість, за яку продавець згоден продати товар без торгів).

Для покупців та учасників торгів пункти 1 и 2 такі ж самі, тільки після цього вони мають можливість приймати участь у торгах аукціону.

Сайт виступає в якості посередника між продавцем та покупцем, надаючи «середовище» для проведення торгів. На більшості таких сайтах, є можливість:

− продивляться фотографії лотів, їх відео-записи;

− читати відгуки та коментарі стосовно певних лотів, осіб, що виставляють лоти (продавців) та осіб, що приймають участь в аукціоні як покупці, а також вести переписку між користувачами для обговорення деталей угоди (форма відправки товару, терміни відправки, форма сплати тощо).

Відповідно до загально прийнятих правил після закінчення торгів продавець та покупець домовляються про спосіб передачі товару та форму сплати. Для цього переможцю надсилаються контактні дані продавця.

Процедура обумовлення способу передачі товарів/грошей може відбуватися як за допомогою сервісів сайту, так й іншими способами – сервіси електронної пошти, миттєвих повідомлень, телефонами тощо.

Більшість з відомих інтернет-порталів мають такий сервіс як «популярність продавця». Суть цього сервісу полягає у тому, що при кожному вдалому продажу товару (товар, що був відправлений покупцеві, належної якості, відповідає описаному у характеристиці лоту та у встановлений домовленістю термін) покупець оцінює продавця за школою довіри та популярності.

До критеріїв, які повинні бути враховані при оцінці продавця, як правило, відносять добросовісність учасника правочину. Під добросовісністю продавця зазвичай розуміють виконання ним своїх зобов’язань, тобто товар, що був відправлений покупцеві, має бути належної якості, відповідати описаному у характеристиці до лоту та маж бути надісланим у встановлений домовленістю термін.

Процедура оцінювання рівня довіри до продавців на різних сайтах може відрізняються, але суть залишається незмінною та зводиться до описаного вище. За весь час діяльності продавця (з моменту реєстрації) складається середньостатистична оцінка його роботи, яка відображається серед інших відомостей стосовно продавця. За загальним правилом, чим більше це число – та більше вдалих угод було укладено, відповідно – більший рівень довіри до продавця.

Аналогічна процедура оцінки рівня довіри використовується стосовно покупців. Таку оцінку проводять, як правило, подавці за подібною методикою, тільки в якості критеріїв оцінки виконання зобов’язань покупця виступає вчасність та відповідність суми оплата заздалегідь домовленим способом. Інформація стосовно рівня довіри до покупця також може відображатися у відомостях про нього.

Така процедура була розроблена для користувачів Інтернет-порталів з метою інформування них про сумлінність продавців та покупців.

Однак, як вже зазначалося вище, ідеї, які спрямовані на поліпшення та покрашення рівня надаваних послуг, зловмисники за частую використовують в своїх злочинних планах. шахраї, як правило, намагаються з початку підняти рівень довіри до себе. Для досягнення цієї мети вони сумлінно виконують свої обов’язки як продавців. Коли досягається достатньо високий рівні довіри, зловмисники починають займатися шахрайствами [9].

Знаючи це тепер можна створити схему за якою відбувається передача грошей/товару в мережі Інтернет (Рисунок 2.1).

Рисунок 2.1 – Схема передачі грошей/товару в мережі Інтернет

З цієї схеми можна виділити декілька випадки здійснення угоди:

1. продавець відправив товар після чого отримав гроші;
2. покупець відправив гроші після чого продавець відправив товар;
3. покупець відправив частину грошей після чого йому був відправлений товар і вже після отримання товару відсилає решту коштів.

Аналізуючи схему, можно дійти висновку, що дії, які вчиняються при шахрайствах передачі товарів/грошей в Інтернеті, можна умовно поділити на два «сценарії»:

– за першим – покупець сплачує вартість лоту, але отримує товар поганої якості (товар не відповідає описаному у характеристиці до лоту за кількісно-якісними характеристиками), отримає не той товар або зовсім його не отримує – умовно назвемо «шахрайства з боку продавця»;

– за другим – продавець висилає товар, а кошти за нього не отримує (повністю, або частково) – умовно назвемо «шахрайства з боку покупця». При шахрайстві з боку покупця – при отриманні товару особа не сплачує вартість товару повністю або частково (Рисунок 2.2).

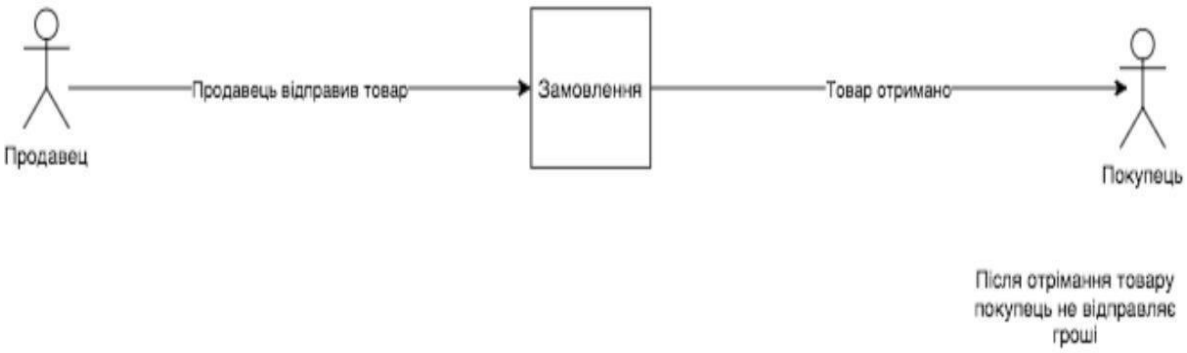

Рисунок 2.2 – Шахрайство з боку покупця

При шахрайствах з боку продавця, враховуючи те , що описання до лоту, що виставляється на торги, робиться самостійно особою, що його виставляє (продавцем), як вже зазначалось вище, зловмисник свідомо та навмисно вносить в його описання такі характеристики, що не відповідають дійсності (кількість, якість, матеріали виготовлення тощо).

Також слід відмітити, що не рідко, визначивши спосіб оплати і отримання грошей шахраї просто не висилають товар зовсім, чи висилають той товар, що не відповідає лоту (Рисунок 2.3).

Рисунок 2.3 – Отриманий товар не відповідає заявленому і товар не відправлений

Отже, при здійсненні операції передачі грошей/товару в мережі Інтернет, ми виділили три точки де можуть виникнути ризики.

Для вирішення проблеми виникнення цих ризиків, ми пропонуємо сторонній сервіс який буде виконувати роль посередника в цих угодах. Цей сервіс має допомогти в передачі грошей чи товару до користувачів без ризиків.

У разі виникнення шахрайства зі сторони покупця чи продавця, він допоможе вирішити суперечку вибравши сторону постраждалого від шахрайства, якщо у цієї сторони будуть вагомі докази.

Під час використання стороннього сервісу, в початковій схемі передачі грошей/товару в мережі Інтернет, (Рисунок 2.1) дещо зміниться. Тепер в схемі бере участь посередник і угода тепер має більше стадій.

Нові стадії і яку саме роль для уникнення ризиків буде відігравати посередник і на яких саме етапах угоди передачі грошей/товару, ми можемо побачити на схемі (Рисунок 2.4).

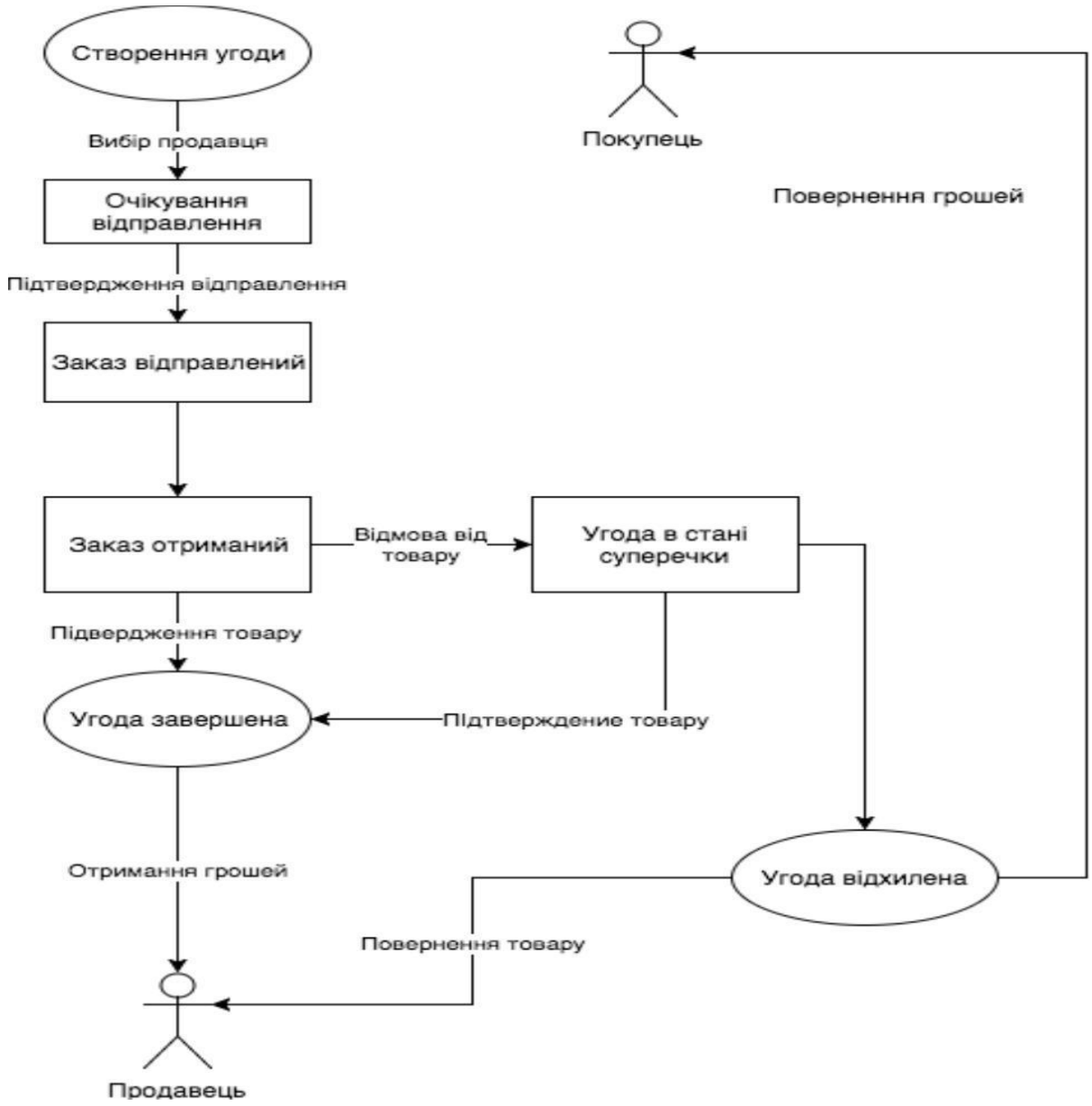

Рисунок 2.4 – Здійснення угоди з посередником

Тепер зупинимось на кожному етапі і розглянемо його більш детально, для більшого розуміння, як саме посередник буде допомагати в здійснені угоди.

Етап перший – після того як покупець і продавець домовилися про конкретний товар і спосіб оплати покупець створює угоду на сайті цього сервісу.

Етап другий – сервіс знімає заявлену суму грошей с карти покупця і зберігає у себе до отримання заказу покупцем.

Етап третій – продавець бачить, що гроші вже є на сервісі і відправляє

товар. Продавець відмічає все це на сервісі додавши накладну.

Етап четвертий – покупець отримує товар, підтверджує отримання і його стан на сервісі. Після чого угода буде завершена і гроші будуть перераховані сервісом на карту продавця.

Це було описано як все має працювати при сумлінній роботі продавця і покупця тобто без шахрайства їх сторін.

Тепер проговоримо стадії на яких можуть виникнути проблеми. На третьому етапі продавець відправляє поганий товар, після чого покупець створює суперечку на сервісі в якій приймають учать покупець, продавець і модератор зі сторони сервісу. Продавець може наприклад додати до суперечки відео на якому розпаковує посилку і де видно, що це не заявлений товар. Коли модератор проаналізує відео, то гроші можуть бути повернуті до покупця. Вирішити суперечку в сторону покупця може і сам продавець, коли визнає свою помилку, що випадково відправив не той товар.

Випадок коли покупець не відправляє гроші не виникає тому, що угоду на сайті сервісу створює сам покупець, після чого у нього відразу знімаються кошти.

Підводячи висновок, можна сказати, що вже є багато наукових робіт в яких досліджували різні види шахрайства в мережі Інтернет. Але більшість з досліджують саме етапи, коли може виникнути ризик шахрайства і пошук слідів цього злочину. Проте ніхто не розкривав тему, як все ж таки здійснювати фінансові операції в мережі Інтернет безпечно. Тому ми постарались розкрити саме цю тему і запропонували вирішення проблеми шахрайства під час передачі грошей/товару. Використовуючи сервіс, який відіграє роль посередника в здійснені фінансової операції. Зменшується ризик виникнення шахрайства, як зі сторони продавця, так і зі сторони покупця.

Цей сервіс реалізовано у вигляді web-додатку. Кожен с учасників угоди повинен мати аккаунт на цьому сайті для переказу або отримання грошей. В web-додатку буде реалізовано зручний пошук користувачів, кому потрібно перерахувати і яку саме суму грошей. Для зручності на головній панелі буде

виведені всі фінансові операції с повною інформацією (відправник грошей, одержувач грошей, дата створення угоди, поточний статус угоди). Також на цій панелі будуть додані кнопки в залежності від статусу угоди. За допомогою яких користувачі зможуть відмінити угоду, підтвердити доставку товару та відкрити суперечку.

Ще однією с головних умов для відправника грошей буде прив’язка кредитної карти до сервісу для зняття грошей для перерахування, а для одержувача це аккаунт в платіжній системі stripe для безпечних транзакцій.

Stripe — американская технологическая компания, разрабатывающая решения для приёма и обработки электронных платежей з дуже великою клиентскою базою.

2.2 Дослідження існуючої інформаційної архітектури вирішення задачі підвищення безпеки

Для створення сервісу підвищення безпеки інформації нами були досліджені такі архітектури іноформаційних систем:

1. файл-сервер
2. виділений сервер бази даних
3. активний сервер баз даних
4. сервер додадків
5. web-додатків

Архітектура «файл-сервер» – організація і управління базою даних (в т.ч. і СУБД) цілком розташовується на машині клієнта, а база даних, що представляє собою зазвичай набір спеціалізованих структурованих файлів, на машині-сервері. В цьому випадку серверна компонента представлена навіть не засобами СУБД, а мережевими складовими операційної системи, що забезпечують віддалений розподілений доступ до файлів.

Взаємодія між клієнтом і сервером відбувається на рівні команд вводу-виводу файлової системи, яка повертає запис або блок даних.

Переваги: можливість обслуговування запитів декількох клієнтів. Недоліки:

- високе завантаження мережі і машин-клієнтів, тому що обмін йде на рівні одиниць інформації файлової системи - фізичних записів, блоків або навіть файлів, з яких на машині клієнта будуть обрані і представлені необхідні для додатка елементи даних;
- низький рівень захисту даних, так як доступ до файлів БД управляється загальними засобами ОС-сервера;
- низький рівень управління цілісністю і непротиворечивостью інформації, так як бізнес-правила функціональної обробки, зосереджені на клієнтської частини, можуть бути суперечливими і несинхронізованих.

Архітектура «виділений сервер бази даних» – засоби управління базою даних і база даних розміщені на машині-сервері.

Взаємодія між клієнтом і сервером відбувається на рівні команд мови маніпулювання даними СУБД (зазвичай SQL), які обробляються СУБД на машині-сервері.

Переваги:

- можливість обслуговування запитів декількох клієнтів;
- зниження навантаження на мережу і машини сервера і клієнтів;
- захист даних здійснюється засобами СУБД, що дозволяє блокувати недозволені користувачеві дії;
- сервер реалізує управління транзакціями і може блокувати спроби одночасного зміни одних і тих же записів.

Недоліки:

- бізнес-логіка функціональної обробки і представлення даних можуть бути однаковими для кількох клієнтських додатків, і це збільшить сукупні потреби в ресурсах при виконанні внаслідок повторення частини коду програм і запитів;

– низький рівень управління непротиворечивостью інформації, так як бізнес-правила функціональної обробки, зосереджені на клієнтської частини, можуть бути суперечливими.

Архітектура «активний сервер баз даних» — для того, щоб усунути недоліки, властиві архітектурі сервера бази даних, необхідно, щоб несуперечливість бізнес-логіки і зміни бази даних контролювалися на стороні сервера.

Для цієї функції бізнес-логіки поділяються між клієнтської і серверної частинами. Загальні або критично значимі функції оформляються у вигляді збережених процедур, що включаються до складу бази даних. Крім цього вводиться механізм відстеження подій БД. Тригери, також включаються до складу бази. При виникненні відповідної події (зазвичай зміни даних), СУБД викликає для виконання збережену процедуру, пов'язану з тригером, що дозволяє ефектно контролювати зміну бази даних.

Збережені процедури і тригери можуть бути використані будь-якими клієнтськими додатками, що працюють з базою даних. Це знижує дублювання програмних кодів і виключає необхідність компіляції кожного запиту.

Недоліком такої архітектури стає істотно зросла завантаження сервера за рахунок необхідності відстеження подій і виконання частини бізнес-правил.

Архітектура «сервер додатків» — розглянуті вище архітектури є дворівневі, але тут всі функції доступу і обробки розподілені між програмою клієнта і сервером БД.

Подальше зниження вимог до ресурсів клієнта досягається за рахунок введення проміжної ланки - сервера додатків, на який переноситься значна частина програмних компонентів управління даними і велика частина бізнес-логіки. При цьому сервери баз даних забезпечують виключно функції СУБД з ведення та обслуговування бази даних.

До інших переваг трирівненвої архітектури можна віднести:

– централізоване ведення бізнес-логіки, і в разі внесення зміни відсутність необхідності їх тиражування в клієнтських додатках;

− відсутність необхідності встановлювати на клієнтських машинах компоненту програмного забезпечення управління доступу до даних:

− можливість відкладеного поновлення БД в разі зміни даних, запитаних з сервера, в автономному режимі. Дані будуть оновлені в базі після наступного з'єднання клієнтської програми з сервером додатків.

Архітектура web-додатків або архітектура web-сервісів має на увазі надання деякого сервісу, доступного в мережі Internet, через спеціальний додаток.

Основою для надання таких послуг служать відкриті стандарти й протоколи SOAP, UDDI й WSDL. SOAP (Simple Object Access Protocol) визначає формат запитів до web-сервісів. Дані між клієнтом і сервісом передаються в SOAP-конвертах (envelops). WSDL (Web Service Description Language) служить для опису інтерфейсу надаваного сервісу. Перед розгортанням web-додатка потрібно скласти його опис, указати адреса, список підтримуваних протоколів, перелік припустимих операцій, а так само формати запитів і відповідей. UDDI (Universal Description, Discovery and Integration) являє собою протокол пошуку web-сервісів у мережі Internet. Пошук здійснюється за їхніми описами, які розташовані в спеціальному реєстрі.

Архітектура таких сервісів схожа за концепцією з багатоланкової клієнт-серверною, однак, сервера додатків і баз даних розташовуються в мережі Internet.

Можна виділити три технології, які можливо використати для побудови розподіленої архітектури web-сервісу:

− EJB (Enterprise JavaBeans);

− DCOM (Distributed Component Object Model);

− CORBA (The Common Object Request Broker Architecture).

Ідеєю для створення EJB було бажання створити інфраструктуру для легкого додавання й видалення компонентів зі зміною функціональності сервера. EJB дозволяє розроблювачам створювати власні додатки із заздалегідь створених модулів. При цьому можлива їхня зміна, що робить

процес розробки гнучким і набагато більше швидким. Дана технологія сумісна з CORBA й Java API.[16]

Взаємодія між клієнтів і сервером у цьому випадку представляється як взаємодія EJB-об’єкта, що генерується спеціальним генератором, і EJB-компонента, написаного розроблювачем. При необхідності викликати метод в EJB-компонента, що перебуває на сервері, викликається однойменний метод EJB-об’єкта, розташованого на стороні клієнта, що зв’язується з необхідним компонентом і викликає необхідний метод.

Переваги EJB:

- просте й швидке створення;
- Java-оптимізація;
- кросплатформність;
- вбудована безпека.

Недоліки EJB:

- складність інтегрування з додатками;
- погана масштабованість;
- низька продуктивність;
- відсутність міжнародної стандартизації.

DCOM являє собою розподілену програмну архітектуру від компанії Microsoft. З її допомогою програмний компонент одного комп’ютера може передавати повідомлення програмному компоненту іншого комп’ютера, причому з’єднання встановлюється автоматично. Для надійної роботи потрібно забезпечити захищене з’єднання між зв’язаними компонентами, а також створити систему перерозподіл трафіку.

Переваги DCOM:

- незалежність від мови;
- динамічне знаходження об’єктів;
- масштабованість;
- відкритий стандарт.

Недоліки DCOM:

- складність реалізації;
- залежність від платформи;
- пошук через службу Active Directory;
- відсутність іменування сервісів через URL.

Технологія CORBA розглядає всі додатки в розподіленій системі як набір об’єктів. Об’єкти можуть одночасно виступати в ролі клієнта й сервера, викликаючи методи інших об’єктів і відповідаючи на їхні виклики. Застосування даної технології дозволяє будувати системи, що перевершують по складності й гнучкості системи з архітектурою клієнт-сервер (як дворівневої, так і трирівневої).[16]

Переваги CORBA:

- незалежність від платформи;
- незалежність від мови;
- динамічні виклики;
- динамічне виявлення об’єктів;
- масштабованість;
- індустріальна підтримка.

Недоліки:

- відсутність іменування по URL;
- практично повна відсутність реалізації CORBA-сервісів;

«Клієнт - сервер» — обчислювальна або мережева архітектура, завдання якої розподіляти навантаження між постачальниками послуг (серверами) і замовниками послуг (клієнтами). За допомогою цієї концепції ми можемо робити різні дії в мережі Intrenet. Клієнт і сервер фізично представленні програмами. Наприклад, типовим клієнтом є браузер, а сервером можна назвати всі HTTP сервера, MySQL сервер та інші.

Клієнт і сервер взаємодіють в мережі Інтернет або в інших комп'ютерних мережах. За допомогою різних мережевих протоколів, наприклад, IP протокол, HTTP протокол, FTP та інші. Таких протоколів дуже багато і кожен протокол дозволяє надати ту чи іншу послугу. За допомогою HTTP протоколу браузер

відправляє спеціальні HTTP повідомлення, в якому вказано яку інформацію і в якому вигляді отримати від сервера. Отримавши таке повідомлення, відсилає браузеру у відповідь схоже по структурі повідомлення в якому міститься потрібна інформація, як правило це HTML документ.[21]

Також варто зауважити, що в основі взаємодії клієнт-сервер лежить принцип того, що таку взаємодія починає клієнт, сервер лише відповідає клієнту і повідомляє про те чи може він надати послугу клієнтові і якщо може, то на яких умовах. Клієнтське програмне забезпечення та серверне програмне забезпечення зазвичай встановлено на різних машинах, але також вони можуть працювати і на одному комп'ютері.

Дана концепція взаємодії була розроблена в першу чергу для того, щоб розділити навантаження між учасниками процесу обміну інформацією, а також для того, щоб розділити програмний код постачальника і замовника.

До одного сервера може звертатися відразу кілька клієнтів (на одному сайті може перебувати кілька відвідувачів). Також варто зауважити, що кількість клієнтів, які можуть одночасно взаємодіяти з сервером залежить від потужності сервера і від того, що хоче отримати клієнт від сервера.

Багато мережних протоколи побудовані на архітектурі клієнт-сервер, тому в їх основі зазвичай лежать однакові або схожі принципи взаємодії, а різницю ми бачимо лише в деталях, які обумовлені особливостями і специфікою області, для якої розроблявся той чи інший мережевий протокол.

Існує два види архітектури взаємодії клієнт-сервер: перший отримав назву дворівнева архітектура клієнт-серверної взаємодії, другий - багаторівнева архітектура клієнт-сервер (іноді його називають трирівнева архітектура або трирівнева архітектура, але це окремий випадок).

Принцип роботи дворівневої архітектури взаємодії клієнт-сервер полягає в тому, що обробка запиту відбувається на одній машині без використання сторонніх ресурсів. Дворівнева архітектура пред'являє жорсткі вимоги до продуктивності сервера, але в той же час є дуже надійною (Рисунок 2.1).

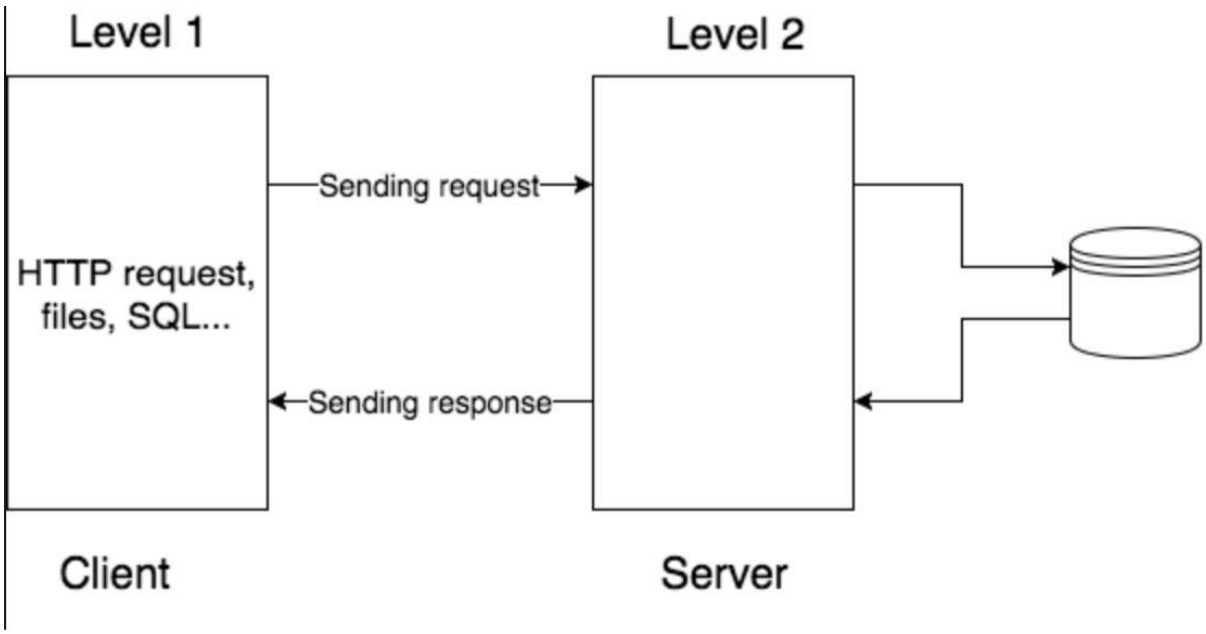

Рисунок 2.1 – Дворівнева модель взаємодії клієнт-сервер

Тут чітко видно, що є клієнт (1-й рівень), який дозволяє людині зробити запит, і є сервер, який обробляє запит клієнта.

Якщо говорити про багаторівневу архітектуру взаємодії клієнт-сервер, то як приклад можна привести будь-яку сучасну СУБД, за винятком, напевно, бібліотеки SQLite, яка в принципі не використовує концепцію клієнт-сервер (Рисунок 2.2).

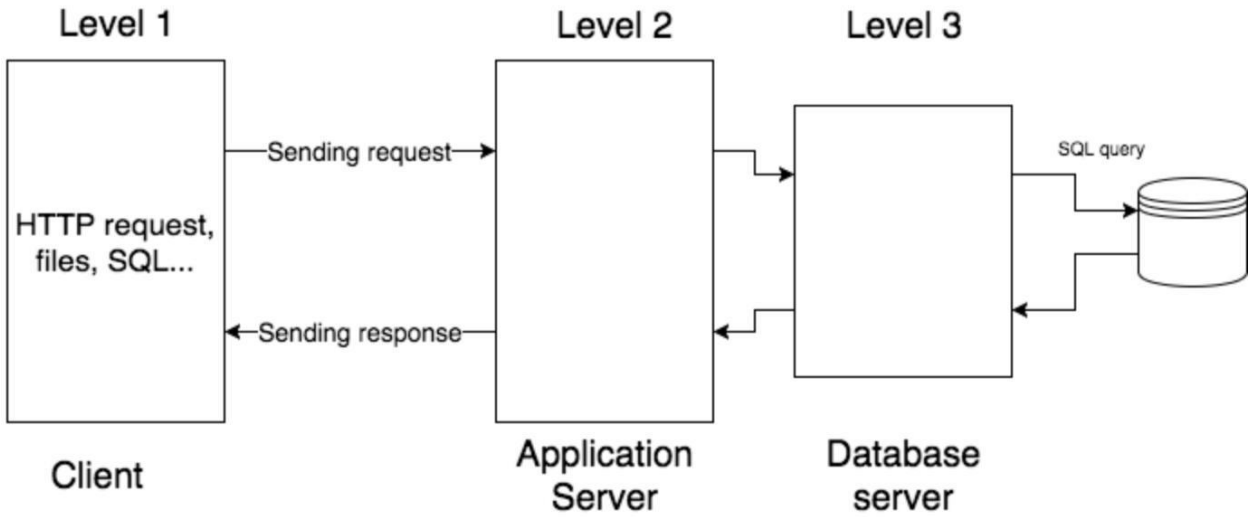

Рисунок 2.2 – Багаторівнева архітектура клієнт-сервер

Суть багаторівневої архітектури полягає в тому, що запит клієнта обробляється відразу декількома серверами. Такий підхід дозволяє значно знизити навантаження на сервер через те, що відбувається розподіл операцій, але в той же самий час даний підхід не такий надійний, як дворівнева архітектура.

Типовий приклад трирівневої моделі клієнт-сервер. Якщо говорити в контексті систем управління базами даних, то перший рівень – це клієнт, який дозволяє нам писати різні SQL запити до бази даних. Другий рівень – це СУБД, який інтерпретує запити і реалізує взаємодію між клієнтом і файлової системою, а третій рівень – це сховище даних.

Якщо ми подивимося на цю архітектуру з позиції сайту. Те перший рівень можна вважати браузером, за допомогою якого відвідувач заходить на сайт, другий рівень – це зв'язка сервер, а третій рівень – це база даних.

Перевагою моделі взаємодії клієнт-сервер є те, що програмний код клієнтського додатка і серверного розділений. Якщо ми говоримо про локальні комп'ютерні мережі, то до переваг архітектури клієнт-сервер можна віднести знижені вимоги до машин клієнтів, так як більша частина обчислювальних операцій буде проводитися на сервері, а також архітектура клієнт-сервер досить гнучка і дозволяє адміністратору зробити локальну мережу більш захищеною.

До недоліків моделі взаємодії клієнт-сервер можна віднести те, що вартість серверного обладнання значно вище клієнтського (про те існує безліч облачних сервісів де можна орендувати сервер). Сервер повинен обслуговувати спеціально навчений і підготовлений людина. Якщо в локальній мережі лягає сервер, то і клієнти не зможуть працювати (в якості окремого випадку можна привести приклад: потужності сервера не завжди вистачає, щоб задовольнить запити клієнтів, якщо ви хоч раз працювали з білінговими системами, то розумієте про що я: час очікування відповіді від сервера може бути дуже великим).

В якості висновку нам варто акцентувати увагу на тому, що архітектура клієнт-сервер не ділить машини на тільки клієнт або тільки сервер, а скоріше дозволяє розподілити навантаження і розділити функціонал між клієнтською частиною і серверної. [6]

2.3 Програмне та апаратне забезпечення безпеки фінансових операцій в мережі Інтернет

Для створення нашого web-додатку ми використовували різне технічне і програмне забезпечення. З технічних засобів у нашому розпорядженні був MacBook Pro 13`` 2017 на базі операційної системи macOS High Sierra.

Характеристики даної моделі:

- дисплей Retina з екраном з діагонал’ю 13,3 дюйма з підсвічуванням LED і технологією IPS, роздільна здатність 2560x1600 пікселів (227 пікселів / дюйм)
- двохядерний процесор Intel Core i5 з тактовою частотою 2,3 ГГц (прискорення Turbo Boost до 3,6 ГГц) і пам'яттю eDRAM об'ємом 64 МБ
- вбудований SSD-накопичувач PCIe ємністю 128 ГБ
- 8 ГБ вбудованої пам’яті LPDDR3 2133 МГц
- графічний процесор Intel Iris Plus Graphics 640

MacOS High Sierra (версия 10.13) була представлена в червні 2017 року. У новій версії ОС була істотно модернізована файлова система Apple (APFS); посилена система забезпечення схоронності даних в разі перебоїв в електропостачанні або системних збоїв, а також додана нова вбудована технологія шифрування даних для захисту інформації. Також, є вбудована система підтримки відеоформату HEVC (H.265), є API для Metal 2 - технологія, що підтримує можливості машинного навчання, які використовуються в розпізнаванні мови, комп'ютерному зорі і інших напрямках.[17]

Основне програмне забезпечення яке ми використовували для розробки були JetBrains WebStorm 2017.2 і JetBrains RubyMine 2017.3.

JetBrains WebStorm 2017.2 – інтегроване середовище розробки на JavaScript, CSS & HTML від компанії JetBrains, розроблена на основі платформи IntelliJ IDEA (Рисунок 2.3).

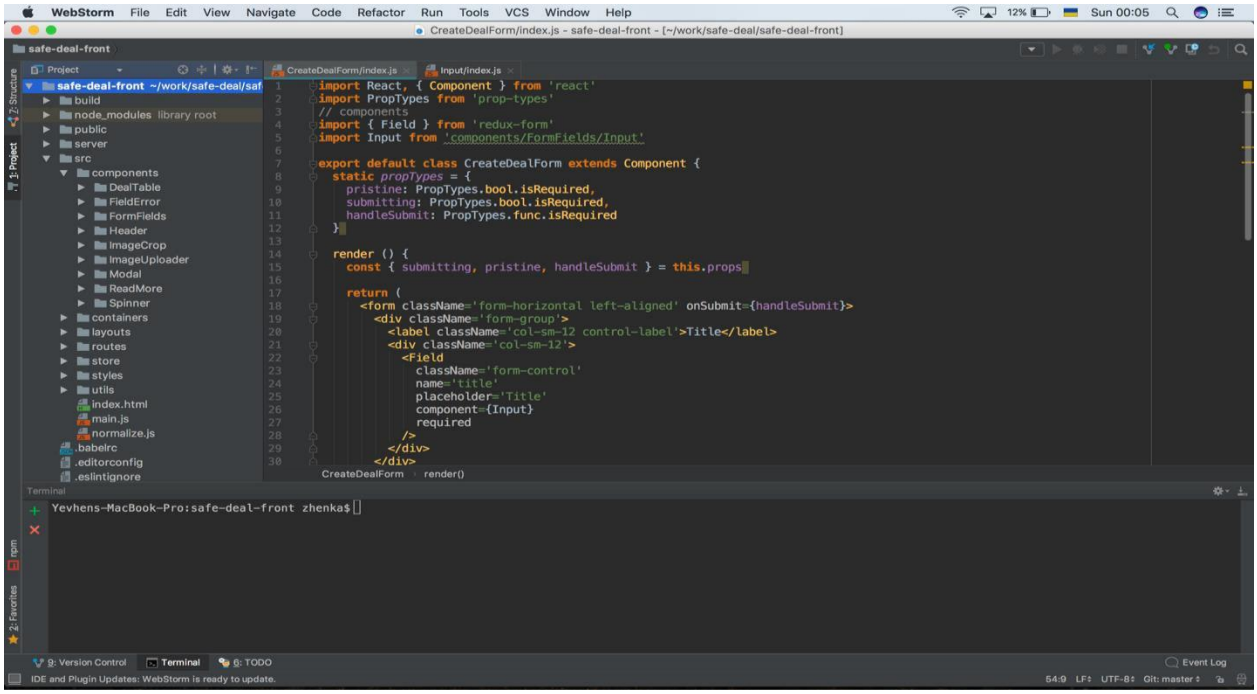

Рисунок 2.3 – Робоче вікно WebStorm 2017.2

WebStorm 2017.2 забезпечує автодоповнення, аналіз коду відразу прі написані, навігацію по коду, рефакторинг, налагодження, і інтеграцію з системами управління версіями. Важливою перевагою інтегрованого середовища розробки WebStorm є робота з проектами (в тому числі, рефакторинг коду JavaScript, що знаходиться в різних файлах і папках проекту, а також вкладеного в HTML). Підтримується множинна вкладеність (коли в документ на HTML вкладений скрипт на Javascript, в який вкладено інший код HTML, всередині якого вкладено Javascript) - тобто в таких конструкціях підтримується коректний рефакторинг. В версії WebStorm 2017.2 були добавлені деякі нові можливості і особливості.

1. Покращення в підтримці JavaScript і TypeScript:

Новий рефакторинг Move symbol переносить класи, глобальні функції і змінні з одного JavaScript або TypeScript файлу в інший. IDE автоматично додасть експорт і оновить ES6-обсяги імпорту в файлах, де використовується цей символ.

Автодоповнення коду і навігація в JavaScript-файлах стали точніше в проектах, що використовують webpack aliases і modules. WebStorm запускає webpack.config.js в корені проекту і використовує результати для побудови більш точної моделі проекту. В результаті WebStorm правильно зрозуміє символи і шляхи в імпорті.

У TypeScript-файлах імена параметрів відображаються прямо в редакторі, щоб було простіше читати код. За замовчуванням, підказки відображаються тільки для параметрів, які є literals або function expressions.

2. Форматування коду:

Якщо ви використовуєте ESLint для перевірки стилю форматування JavaScript-коду, WebStorm запропонує застосувати правила, описані в .eslintrc або в поле eslintConfig в package.json, до проектних налаштувань форматування JavaScript. Коли ви відкриєте JavaScript-файл в такому проекті, ви побачите нотифікацію "Apply code style from ESLint?".

Головна мета цієї інтеграції - зробити так, щоб переформатування коду за допомогою IDE не ламати вірно відформатований код з точки зору ESLint і допомагало писати новий код, який максимально відповідає правилам з .eslintrc.

3. Покращення в підтримці Sass і SCSS:

Автодоповнення імен класів в HTML-файлах тепер працює для Sass- і SCSS-селектор, створених через &.

Крім цього в WebStorm 2017.2 ви можете:

Показувати файли з однаковими іменами, але різними розширеннями з однієї директорії згрупованими. Змінити правила угруповання можна через File nesting в налаштуваннях Project view (іконка з шестерінкою).

Використовувати поліпшену підтримку React stateless-компонентів і автодоповнення props.

Використовувати breadcrumbs внизу редактора в JavaScript- і

Автоматично замінити module.exports на ES6-експорт.

Налаштувати і перевіряти формат повідомлення про комміт в налаштуваннях Version Control | Commit Dialog.

Відкочувати комміт в Git і змінювати кому-повідомлення через контекстне меню коммітов в Logs у вікні Version Control [18].

RubyMine - комерційна IDE для розробки програмного забезпечення на Ruby компанії JetBrains.

RubyMine створений на основі IntelliJ IDEA того ж виробника. RubyMine забезпечує інтелектуальне доповнення сирцевого коду Ruby та Ruby on Rails code, аналіз коду на льоту та підтримку рефакторингу для проектів Ruby та веб-застосунків, побудованих з Ruby on Rails.

Підтримує популярні бібліотеки, які використовуються в Ruby-додатках, в тому числі Bundler, RSpec, Shoulda, Cucumber, Git (Рисунок 2.4).

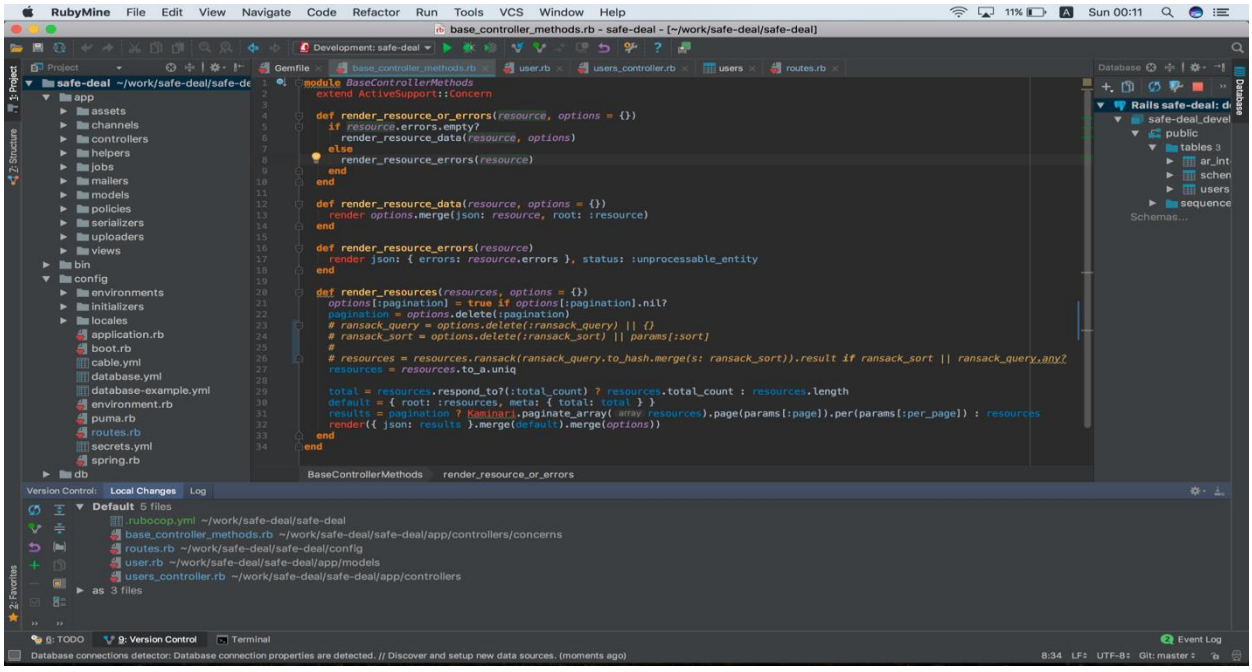

Рисунок 2.4 – Робоче вікно RubyMine 2017.3

В першу чергу слід відзначити продуктивність. Це особливо помітно при швидкості обробки великого об'єму завдань - наприклад, тестуванні роботи всієї програми. Покращена орієнтація у вбудованих ключових словах Rails,
таких як колбеки before_action. Вдосконалений аналізатор коду краще справляється з маппінгом зазначених і заданих аргументів методів. Крім того, нова RubyMine 2017.3 тепер підтримує програми з вкладеними проектами: вона виявляє розташовані в додатку проекти і їх gem-файли, і дозволяє просто вставляти їх в код і налаштовувати їх параметри. Також в версії 2017.3 були додані такі можливості.

В цій версії є підтримка Windows Bash, а також є можливість установки WSL.

Зміни торкнулися також рефакторингу. У цій версії IDE можна витягувати методи прямо в розділи private або protected.

Користувач може не тільки призвести автокорекцію всього файлу, а й створити особливий cop department або offence class. Більш того, RubyMine 2017.3 підтримує призначені для користувача конфігурації rubocop.yml, що робить редагування коду ще зручніше.

Доопрацьовано управління стилем коду - є можливість індентіровать приватні і захищені методи, а також вибирати, навколо яких операторів потрібно додавати прогалини, а які залишати без змін.

Ще одним нововеденням RubyMine 2017.3 є можливість тестувати API в IDE.

Були доопрацьовані функції автозавершення коду і виведення інформації для стандартних об'єктів, методів і Web API JavaScript.

Підтримка Vue.js і функції завершення і переходу до визначення тепер працюють із вхідними параметрами Vue, властивостями в об'єктах даних, що обчислюються властивостями, а також методами.

Поліпшено роботу з CSS: завдяки новим варіантам закінчень коду для значень з властивостями transform, transition і pointer-events підвищилася точність роботи функції автозавершення коду для властивостей і значень CSS.

В цій версії було вдосконалено систему контролю версій: так, в функцію перегляду подій додано дію Interactively Rebase from Here, що дозволяє підбирати, редагувати, пропускати, стискати, переформулювати код і фіксувати його зміни. Зберігаються настройки кожної області робочого простору: відкриті файли, поточну конфігурацію і всі контрольні точки. Також
IDE показує файли, які були змінені коммітом злиття, навіть якщо вони відрізняються від загального батька.

Доопрацювання зазнали бази даних: вузли схем в дереві бази даних були замінені функцією контекстного меню, що показує кількість схем, що відображаються на дереві в певний момент часу. Для створення DDL-скриптів тепер можна використовувати SQL-генератор. Крім того, були поліпшені інструменти роботи з SSH [19].

Для контролю версій нашого проекту ми використовували Git.

Git (вим. «Гіт» [6]) - розподілена система керування версіями. Проект був створений Лінус Торвальдс для управління розробкою ядра Linux, перша версія випущена 7 квітня 2005 року. На сьогоднішній день його підтримує Джун Хама.

Він синхронізує роботу з сайтом і зберігає / оновлює версії файлів, що дуже зручно якщо над проектом працюють кілька розробників одночасно. Git дає можливість оновлювати і правити файли проекту з урахуванням змін внесених іншими.

Зараз git використовується в багатьох сучасних проектах: Android, jQuery, php, Drupal, Wine, Chromium деякі версії Linux і т.д.

Можно перерахувати такі переваги git перед іншими DVCS:

Висока продуктивність.

Розвинені засоби інтеграції з іншими VCS, зокрема, з CVS, SVN і Mercurial. Крім різноспрямованих конвертерів репозиторіїв, наявні в комплекті програмні засоби дозволяють розробникам використовувати git при розміщенні центрального сховища в SVN або CVS, крім того, git може

імітувати cvs-сервер, забезпечуючи роботу через клієнтські програми і підтримку в середовищах розробки, спеціально що не підтримують git.

Продумана система команд, що дозволяє зручно вбудовувати git в скрипти.

Репозиторії git можуть поширюватися і оновлюватися загальносистемними файловими утилітами архівації та відновлення, такими як rsync, завдяки тому, що фіксації змін і синхронізації не змінюють існуючі файли з даними, а тільки додають нові (за винятком деяких службових файлів, які можуть бути автоматично оновлені за допомогою наявних в складі системи утиліт). Для роздачі сховища по мережі досить будь-якого веб-сервера [20].

Для створення бази даних ми встановили СУБД PostgreSQL. Реляційні і об'єктно-реляційні СУБД є одними з найпоширеніших систем. Вони являють собою таблиці, у яких кожен стовпчик, який називається полем, впорядкований і має певну унікальну назву. Послідовність рядків, які називаються записами визначається послідовністю введення інформації в таблицю. При цьому оброблення стовпців і рядків може відбуватися в будь-якому порядку. Таблиці з даними пов'язані між собою спеціальними зв’язками, завдяки чому з даними з різних таблиць можна працювати, наприклад об'єднувати їх за допомогою одного запиту.

При створенні структури таблиці кожне поле запису повинне містити заздалегідь описаний тип. Всі СУБД мають в своєму складі різні типи даних, які не завжди є взаємозамінними. При роботі з СУБД завжди доводиться стикатися з подібними обмеженнями.

Для керування базами даних застосовується особлива мова програмування - SQL. Скорочення розшифровується як "Structured query language", в перекладі на українську «мова структурованих запитів».

Команди, які використовуються в SQL, діляться на ті, які маніпулюють даними, ті, які визначають дані, і ті, які керують даними.

Схема роботи з базою даних виглядає наступним чином рисунок 2.5.

Рисунок 2.5 – Схема роботи з базою даних

PostgreSQL є самою професійною з усіх трьох розглянутих СУБД. Вона вільно розповсюджується і максимально відповідає стандартам SQL.

Від інших СУБД PostgreSQL відрізняється підтримкою затребуваного об'єктно-орієнтованого і реляційного підходу до баз даних. Наприклад, повна підтримка надійних транзакцій, тобто атомарність, узгодженість, ізольованість, надійність. Завдяки потужним технологіям Postgre дуже продуктивна. PostgreSQL дуже легко розширювати своїми процедурами, які називаються збережені процедури. Ці функції спрощують використання постійно повторюваних операцій.

Переваги PostgreSQL:

- відкрите ПЗ відповідає стандарту SQL. Ця СУБД є дуже потужною системою;
- велика кількість доповнень - незважаючи на величезну кількість вбудованих функцій, існує дуже багато доповнень, що дозволяють розробляти дані для цієї СУБД і управляти ними;
- розширення - існує можливість розширення функціоналу за рахунок збереження своїх процедур;
- об'єктність - PostrgreSQL це не тільки реляційна СУБД, але також і об'єктно-орієнтована з підтримкою успадкування і багато іншого

Недоліки PostgreSQL:

- продуктивність - при простих операціях читання PostgreSQL може значно уповільнити сервер і бути повільніше своїх конкурентів, таких як MySQL;

- популярність - за своєю природою, популярністю ця СУБД похвалитися не може, хоча і є досить велика спільнота;
- хостинг - в силу названих вище чинників іноді досить складно знайти хостинг з підтримкою цієї СУБД [6].

В результаті стає зрозуміло, що MySQL підходить для тих випадків, де необхідна СУБД для проекту невеликого або середнього розміру, швидка і зручна в роботі і без складнощів з адмініструванням. Саме тому дана СУБД найкраще підходить для даного проекту.

3 РЕАЛІЗАЦІЯ ПРОТОТИПУАВТОМАТИЗОВАНОЇ СИСТЕМИ ЕКОНОМІЧНОГО ОБ'ЄКТА

3.1 Створення прототипу інтерфейсу та back-end автоматизованої системи

Розробку додатку, ми почали зі створення прототипу. Прототип – це детальний вигляд веб-сторінки. Він будується для того, щоби продумати інтерфейс перед тим як розпочати створювати його дизайн. Добре створений прототип є повноцінним каркасом сайту.

Для створення прототипу ми використали сервіс "Mockingbot". Ми вибрали його через те що він має дуже легкий і зрозумілий інтерфейс і не потрібно бути професіональним дизайнером чи проектним менеджером для того щоб їм користуватись. Є можливість вибирати для якого саме пристрою створювати каркас і в залежності від цього буте надана можливість настройки розмірів шаблону, також присутня велика кількість уже створених шаблонів з можливістю їх модифікування. Ще однією з переваг є великий набір компонент таких як кнопки, перемикачі, випадаючий список та іконки. Можна налаштувати перехід між уже створеними сторінками тобто відтворити поведінку натискання на кнопку і переходу на іншу сторінку сайта. Після створення прототипу є можливість його перегляду в режимі тестування і за одно продивитися як будуть себе поводити елементи підчас зміни розмірів екрану.

Перед створенням додатку ми продумали такий інтерфейс нашої основної сторінки (Рисунок 3.1).

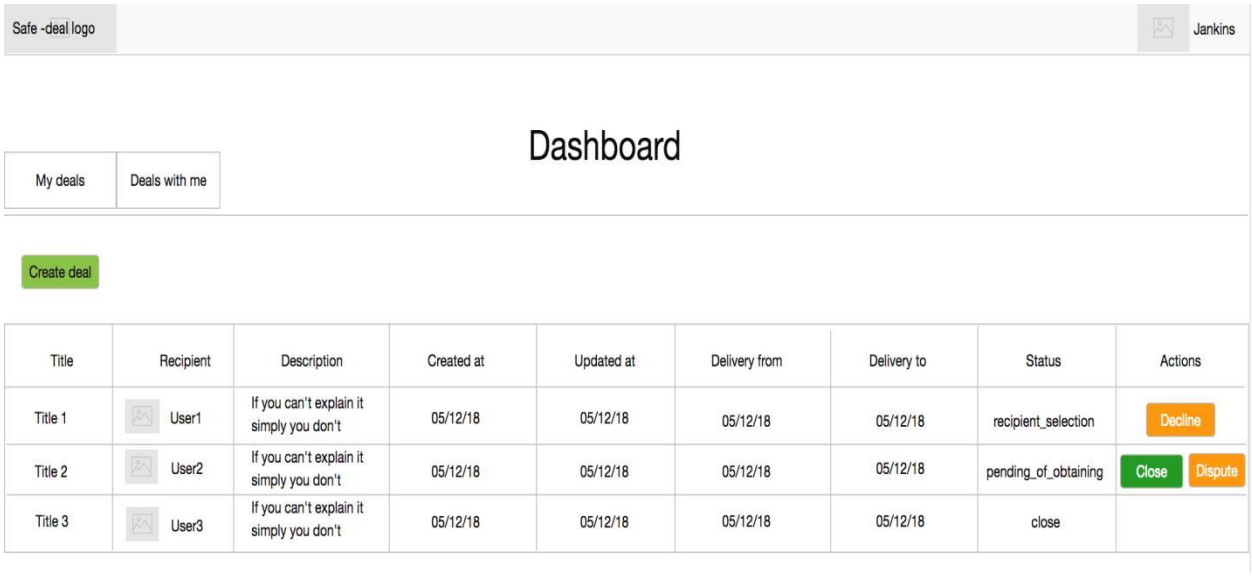

Рисунок 3.1 – Прототип основної сторінки

На нашому прототипі зображено шапка сайту с логотипом та іконка користувача з випадаючим списком. Цей список містить в собі посилання на інші сторінки сайту. Нижче є заголовок сторінки і два таби. Таб “My deals” містить у собі таблицю с угодами які створив сам користувач. На другому табі “Deals with me” знаходиться таблиця с угодами які створили інші користувачі зі мною.

Також на цій сторінці є кнопка яка відкриваю модальне вікно в якому знаходиться форма для створення угоди.

Після побудови прототипу, нами було створено back-end проекту с фреймворком Ruby on Rails. Цей фреймворк написаний на мові програмування ruby і за допомогою нього дуже легко і швидко писати API для веб-додатку.

Спершу створюємо таблицю користувача за допомогою генерації міграції. Міграція – це швидкий спосіб змін бази даних без написання SQL запитів. Лістинг коду приведено в додатку Б.

Далі для автентифікації користувача в нашій системі ми підключаємо два геми: devise і devise_token_auth [26]. За допомогою їх наш користувач буде отримувати токен завдяки якому зможе авторизуватися на нашому сервісі і звертатися до нашої API поки не закінчиться сесія.

Переходимо до налаштування devise_token_auth. В routes.rb було добавлено

роути до яких будемо звертатися для реєстрації, авторизації, і зміни паролю іх приведено на лістингу 3.1. Весь файл routes.rb можно переглянути в додатку В.

Лістинг 3.1 – Кастомні роути для devise_token_auth

```
mount_devise_token_auth_for 'User', at: 'auth', skip: [:omniauth_callbacks], controllers:
  {
    registrations: 'auth/registrations',
    sessions: 'auth/sessions',
    token_validations: 'auth/token_validations',
    passwords: 'auth/passwords'
  }
```

Кожен з цих роутів ссилається на свій контролер який в свою чергу взаємодіє з моделлю і повертає результат для його зображення. Нижче на лістингу 3.2 приведемо приклад sessions_controller.rb інші контролери можна переглянути в додатку Г.

Лістинг 3.2 – Приклад sessions_controller.rb

```
class Auth::SessionsController <
  DeviseTokenAuth::SessionsController def render_create_success
    render_resource_data(@resource)
  end
end
```

В контролерisessions_controller.rbмиперезаписалиметод render_create_success який спрацьовує коли сесія успішно створена.

Контролер registrations_controller.rb зазнав зміну таких методів як render_create_success, render_create_error, render_update_success,

render_update_error. З назв цих методі зрозуміло коли саме вони визиваються.

Контролер token_validations.rb відповідає за перевірку токена на валідність. В ньому зазнав змін метод render_validate_token_success. І в контролері passwords_controller.rb зазнав змін метод який спрацьовує при успішній зміні паролю render_create_success.

Кожен с цих методів в свою чергу визивав методи для повернення різного виду результату. Вони були записані в окремий модуль base_controller_methods.rb для подальшого використання в інших контролерів. Це дає змогу не дублювати однаковий код в різних місцях. Лістинг цього файлу приведено в додатку Д.

Наступним шагом для налаштування devise_token_auth ми додаємо “include DeviseTokenAuth::Concerns::SetUserByToken” в api_controller.rb. Цей файл є нашим головним контролером. Всі інші контролери будуть унаслідуватись від нього і тому всі його методи будуть в них доступні. Так наприклад модуль base_controller_methods.rb включається в цьому контролері за допомогою такого рядка “include BaseControllerMethods” після чого стає доступних у всіх дочірніх контролерах. Повністю код файлу приведено в додатку Е.

В цьому контролері описані декілька приватних методів. Приватні методи – це методи які доступні тільки в цьому класі.

Методи які представлені на лістингу 3.3 повертають помилки якщо один із запитів не задовольняє деяких умов.

Лістинг 3.3 – Помилки з класу ParameterMissing і RecordNotFound

```
rescue_from ActionController::ParameterMissing do |exception|
  render json: { errors: exception.message }, status:
:bad_request end
rescue_from ActiveRecord::RecordNotFound do |exception|
  render json: { errors: exception.message }, status: :not_found
end
```

ActionController::ParameterMissing повертає помилку коли в запиті не вистачає чи є не правильні поля.

ActiveRecord::RecordNotFound видає помилку коли ви намагаєтесь отримати об’єкта який не був знайдений в базі даних.

Переходимо до іншого етапу налаштування автентифікації користувача. Додаємо декілька захищених методів файл application_controller.rb. (Лістинг 3.4)

Лістинг 3.4 – Дозволені параметри для запитів

```
devise_token_auth def configure_permitted_parameters
  devise_parameter_sanitizer
    .permit(:sign_up, keys: %i[email password
  password_confirmation]) devise_parameter_sanitizer
    .permit(:account_update,
          keys: UserPolicy.new(nil, nil).permitted_attributes)
end
```

Ці методи описують які саме поля можна посилати для реєстрації користувача чи його обновлені. Регулювати коли саме спрацьовувати цей метод буде функція зворотного виклику “before_action :configure_permitted_parameters, if: :devise_controller?” яка спрацьовую тільки перед діями devise контролера. Ввесь файл надано в додатку Ж.

Перейдемо до нашої моделі користувача де опишемо зв’язки які ми описали при будівництві архітектури бази даних і завершимо налаштування гему devise_token_auth.rb. Для цього ми повинні додати рядок “include DeviseTokenAuth::Concerns::User до файлу user.rb”. Лістинг коду надано в додатку З. На цьому налаштування автентифікації користувача завершено.

Далі ми додали можливість додавати та змінювати інформацію про себе. Зміна паролю і контактних даних були налаштовані після підключення гему devise і devise_token_auth. Для зміни аватару потрібно налаштувати додатково

геми carrierwave та minimagic [27, 28]. Список всіх гемів міститься в Gemfile лістинг файлу наданий в додатку И.

Для налаштування гемів які нам дають можливість загружати і змінювати аватар користувача, спершу потрібно створити uploader. Файл base_uploader.rb приведений на лістингу 3.5, унаслідуваний від базового класу CarrierWave::Uploader::Base і містить в собі основні налаштування гему carrierwave такі як де саме зберігати картинку, повний шлях до неї так параметри захищеного токену.

Лістинг 3.5 – Файл base_uploader.rb

```
class BaseUploader <
  CarrierWave::Uploader::Base storage :file
  def store_dir
    "uploads/#{model.class.to_s.underscore}/#{mounted_as}/#{model.id}"
  end
  protected
  def secure_token
    var = :"@#{mounted_as}_secure_token"
    model.instance_variable_get(var) or model.instance_variable_set(var, SecureRandom.hex(8))
  end
end
```

Створюємо ще один файл image_uploader який буде дочірним класом раніше створенного BaseUploader. В якому у нас будуть міститись налаштування гему minimagic і який описує список дозволених форматів картинки і метод який генерує назву збереженого файлу. Також вказаний формат в який конверутвати картинку. Переглянути файл можна на лістингу 3.6.

Лістинг 3.6 – Файл image_uploader.rb

```
class ImageUploader < BaseUploader
  include CarrierWave::MiniMagick
  process convert: 'jpg'
  def extension_white_list
    %w(jpg jpeg gif png)
  end
  def filename
    "image_#{secure_token}.jpg" if original_filename.present?
  end
end
```

Переходимо до створення avatar_uploader.rb (Лістинг 3.7) в якому будуть описані варіанти розмірів картинок які будуть у зберігатись на нашому сервісі.

Лістинг 3.7 – Файл avatar_uploader.rb

```
class AvatarUploader < ImageUploader
  version :medium do
    process resize_to_fill: [200, 200]
  end
  version :small, from_version: :medium do
    process resize_to_fill: [50, 50]
  end
end
```

Для завершення налаштувань зміни аватара користувача додамо рядок “mount_uploader :avatar, AvatarUploader” до моделі user.rb і створимо міграцію яка додасть поле avatar для зберігання шляху до картинки (Лістинг 3.8).

Лістинг 3.8 – Міграциія додаваня поля аватар для користувача

```
class AddImageUrlsToUser < ActiveRecord::Migration[5.1]
```

```
  def change
    add_column :users, :avatar, :string
  end
end
```

Щоб відображати лише потрібні поля таблиць було підключено гем active_model_serializers який буде за це відповідати [29]. Також створимо файл конфігурації active_model_serializers.rb. (Лістинг 3.9)

Лістинг 3.9 – Файл конфіг active_model_serializers.rb

```
ActiveModelSerializers.config.adapter = :json
ActiveModelSerializers.config.default_includes = '**'
```

І створимо файли users_serializer.rb в якому будуть прописані поля які ми повинні відображати. (Лістинг 3.10)

Лістинг 3.10 – Файл users_serializer.rb

```
class UserSerializer < ActiveModel::Serializer
  attributes :id, :email, :name, :phone, :avatar, :verified, :role
end
```

Тепер ми будемо бачити тільки ті поля які прописані в цьому серіалайзері.

Перейдемо до створення наступної моделі Deal. Яка буде зв’язувати двох окремих користувачів і зберігати суму коштів яку потрібно відправити від одного до іншого. Для цього додамо міграцію яка створить модель. (Лістинг 3.11)

Лістинг 3.11 – Міграція створення моделі Deal

```
class CreateDeals < ActiveRecord::Migration[5.1]
```

```
  def change
    create_table :deals do |t|
      t.integer :state
      t.integer :user_id
      t.integer :recipient_id, foreign_key: true
      t.string :title
      t.string :description
      t.float :amount
      t.timestamps
    end
  end
end
```

Код моделі deal.rb наданий в додатку К. Ця модель має два зв’язки з моделлю User. Перша це belongs_to :user і has_one :recipient. Recipient – це також користувач, але він відноситься до моделі як користувач для кого створили цю угоду. Ще для цієї моделі ми підключили гем AASM який буде допомагати змінювати і відстежувати поточний стан угоди [30].

Всього модель Deal має п’ять станів: recipient_selection, pending_of_obtaining, closed, disputed, declined. За допомогою цього гему було створено чотири методи, які можна побачити на додатку З. Ці методи будуть змінювати стан угоди в залежності від поточного і повертати помилку якщо щось не буде задовільняти умову переведення.

Метод decline змінює стан на declined, але за умови що попередні були recipient_selection або disputed.

Метод send_product переводе зі стану recipient_selection в стан pending_of_obtaining.

Метод close – зі станів pending_of_obtaining і disputed в closed&

Метод start_dispute – с стану pending_of_obtaining в disputed і після збереження нового стану запускає метод send_dispute_email який надсилає

кожному з учаснику повідомлення на електрону пошту про те що було розпочато суперечку і також приєднує до неї модератора нашого сервісу, який буде слідкувати за перебігом суперечки і вирішувати кому саме надати перевагу в суперечці. Для відправлення повідомлення ми налаштували SMTP сервер с поштою gmail. SMTP (англ. Simple Mail Transfer Protocol - простий протокол передачі пошти) – це широко використовуваний мережевий протокол, призначений для передачі електронної пошти в мережах TCP / IP.

Для цього ми додали в конфіг файл production.rb налаштування наведені в лістингу 3.14. Повністю файл наведений в додатку Л.

Лістинг 3.14 – SMTP налаштування для gmail.

```
config.action_mailer.smtp_settings = {
   :address              => "smtp.gmail.com",
   :port                 => 587,
   :user_name            => ENV['gmail_username'],
   :password             => ENV['gmail_password'],
   :authentication       => "plain",
   :enable_starttls_auto => true
}
```

Далі було створено клас DisputeMailer який унаслідувався від ApplicationMailer (Лістинг 3.15). І в якому вказані змінні які ми будемо використовувати при заповнені шаблону повідомлення.

Лістинг 3.15 -- Клас DisputeMailer

```
class DisputeMailer < ApplicationMailer
  def dispute_email(deal, recipient, recipient_user)
    @deal = deal
    @recipient = recipient
    @recipient_user = recipient_user
```

```
    mail(to: recipient, subject: 'Start Dispute')
  end
end
```

Далі створимо файл dispute_email.html.erb (Лістинг 3.16) який і буде нашим шаблоном, що буде заповнюватись даними зі змін які ми передаємо сюди і відправлятись до користувачів які відкрили суперечку.

Лістинг 3.16 – Файл шаблон повідомлення dispute_email.html.erb

```
<!DOCTYPE html>
<html>
  <head>
    <meta content='text/html; charset=UTF-8' http-equiv='Content-Type'
  /> </head>
  <body>
    <h1>Hi <%= @recipient %></h1>
    <p>The dispute began on the deal <%= @deal.title %></p>
    <p>Dispute between <%= @deal.user.email %> and <%=
@recipient_user.email %></p>
    <p>Message: <%= @deal.message
  %></p> </body>
</html>
```

Після всіх налаштувань в моделі створімо контролер який буде створювати угоду і змінювати поточний стан угоди. Для цього додамо до файлу routes.rb код приведений в лістингу 3.17.

Лістинг 3.17 – Роути додані в routes.rb

```
    resources :deals, only: %i(index create update) do
      put :decline, on: :member
```

```
    put :send_product, on: :member
    put :start_dispute, on: :member
    put :close, on: :member
  end
```

Після у нас будуть доступні маршрути для виклику таких методів в контролері deals_controller.rb: index, create, decline, send_product, start_dispute, close. Код файлу deals_controller.rb можно переглянути в додатку М.

Далі більш детальніше про методи які ми не описували: index і create. Інші методи були описані коли мова йшла про модель Deal.

Метод create (Лістинг 3.18) створює нову угоду за параметрами які ми передаємо: recipient_id, amount, title, description, email_of_recipient.

Лістинг 3.18 – Метод index контролера

```
deals_controller.rb def create
  authorize Deal
  user = User.find_by_email(params[:resource][:email_of_recipient])
  if user.present?
    deal = current_user.deals.create permitted_attributes(Deal).merge(
recipient_id: user.id)
    render_resource_or_errors(deal)
  else
    render json: { errors: 'This user is not found' }, status:
  :unprocessable_entity end
end
```

Ми знаходимо користувача для якого створюється угода за його email. Якщо користувача с таким email не знайдено буде викликана помилка про те що такого користувача не існує в нашій базі даних.

Метод index (Лістинг 3.19) повертає список всіх угод для кожного

окремого користувача.

Лістинг 3.19 – Метод index контролера

```
deals_contorller def index
  authorize Deal
  render_resources current_user.role == 'admin' ? Deal.where(state: 3) :
    user_deal, each_serializer: deal_serializer
end
```

Для того щоб метод повертав угоди ми додали в модель User рядки які представлені в лістингу 3.20.

Лістинг 3.20 – Зв’язки в моделі User

```
has_many :deals, -> { order('created_at desc') }
has_many :deals_with_me, -> { order('created_at desc') }, class_name: 'Deal',
foreign_key: 'recipient_id'
```

Ці зв’язки повертають різні угоди в залежності від того ким користувач є в цих угодах. Для того щоб відображати угоди по різному нами було створено два різни серіалайзери для с кожного з виду угод (Додаток Н).

Тепер створюємо можливість верифікувати свою карту на нашому сервісі для подальшого розрахунку і отримання на неї коштів. Додаємо рядок “resources :users do put :verify, on: :member end” до файлу routes.rb після чого буде можливість викликати метод verify у контролері users_controller. (Лістинг 3.21)

Лістинг 3.21 – Метод verify контролеру

```
users_controller.rb def verify
  authorize current_user
  current_user.verify_card params[:resource]
```

```
  render_resource_or_errors current_user
end
```

Який в свою чергу після успішної верифікації карти успішно змінить поля користувача, в іншому разі поверне помилку.

3.2 Створення front-end та інструкція з використання

Тепер коли ми написали back-end для наго веб-додатку можна розпочати розробку front-end частини. Розробляти веб-інтерфейс ми вирішили за допомогою React.js. React.js – це бібліотека написана на мові програмування javascript. Ця бібліотека для створення інтерфейсів користувача, яка покликана вирішувати проблеми часткового оновлення вмісту веб-сторінки, з якими стикаються в розробці односторінкових застосунків. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників. Його мета полягає в тому, щоб бути швидким, простим, масштабованим React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні модель-вид-контролер [32]. Також нами був використана бібліотека Redux.js. Redux – це інструмент управління як станом даних, так і станом інтерфейсу в JavaScript-додатках. Він підходить для односторінкових додатків, в яких управління станом може з часом стає складним. Redux не пов'язаний з якимось певним фреймворком тому може бути використаний також с Angular.js [33].

В папці components зберігаються компоненти які використовується декілька разів в проекті.

Containers містить в собі компоненти які є обгортками для компонент с папки components і передають їм якісь значення у властивостях.

В папці routes зберігаються всі компоненти які є нашими сторінками сайту.

Папка static зберігає всі статичні файли проекту.

Store –це папка в якій є файли, що зберігають в собі інформацію для використання її з різних компонент. Тобто тут знаходяться файли які формують глобальний state проекту.

Всі глобальні стилі проекту зберігаються в папці styles

Різні файли с функціями які часто використовуються по всьому проекту поміщають в папку utils. Файл index.html є основним html каркасом де всередині всі інші компоненти і сторінки вирисовуються. В файлі main.js описані функції з яких починається ініціалізація проекту.

Для цього проекту нами було підключено тему “unity” с сайту bootswatch.com. Цей сервіс пропонує набір bootstrap стилів поділених на окремі теми. Їх дуже легко доповнювати і змінювати. Це нам дозволило не витрачати час розробки нашого сервісу на обміркування дизайну нашого сайту [31].

Для початку ми створили шапку нашого сайту так як цей компонент є на кожній із наших сторінок. В шапці є дві кнопки: Login і Register. Також на ньому є випадаючий список зі списком доступних сторінок для користувача (Додаток Т).

Спершу куди потрапляє користувач – це сторінка реєстрації чи авторизації, це залежіть від того чи вже реєструвався цей корстувач. Лістинг коду компоненти, контейнеру і стилів реєстрації наведені в додатку П. На ній знаходиться форма с полями email, password і confirmation password. Користувач повинен їх заповнити і натиснути кнопку Register (Рисунок 3.2).

Щоб на сторінку можливо було перейти потрібно її додати до головного index файлу в папці routes. Повний код index файлу надається в додатку Р.

В цьому файлі містяться всі головні роути нашого сайту. В кожному роуті є дочірні роути де також є подібний файл, для їх дочірниї сторінок.

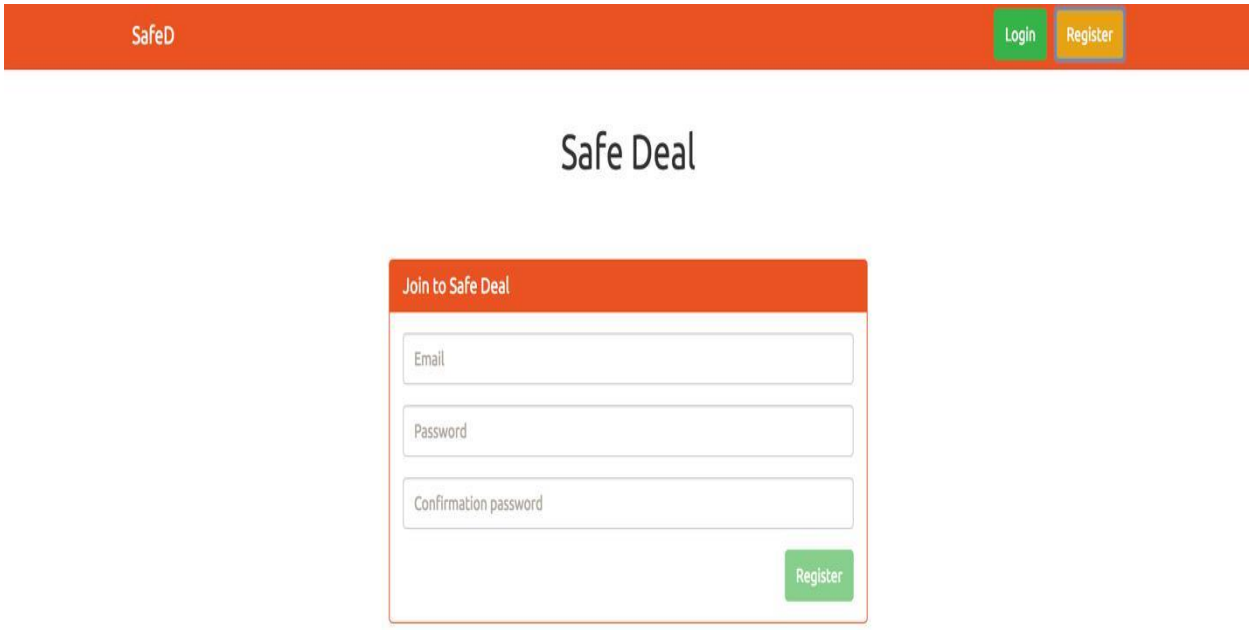

Рисунок 3.2 – Сторінка реєстрації

В index.js паки routes є функція preloadUser, завдання якої при першому заході на головний роут звантажувати дані поточного користувача і якщо він є не авторизованим чи сесія була завершена то повертати його назад на сторінку авторизації чи реєстрації. Після авторизації чи перевірки користувача, його дані зберігаються в редюсері auth. Та забисуються localeStorage. LocaleStorage – це па’ять вашого браузеру. Там також зберігаються деякі дані про вас (email, токен, uid). Ці данні потрібні для першоі ініціалізації (наприклад коли ви перезавантажуєете сторінку вам потрібно повторно авторизуватись), щоб авторизуватись та завантажити актуальні данні про користувача с серверу. Цей редюсер описаний в файлі auth.js де міститься всі дані користувача і всі екшени які його оновлюють новими завантаженими даними. Такі екшени як registration, login, loadCurrentUser, logout та інші. Файл auth.js наведений в додатку С.

Після реєстрації чи авторизації користувач буде переправлений на сторінку власного профайлу (Рисунок 3.3).

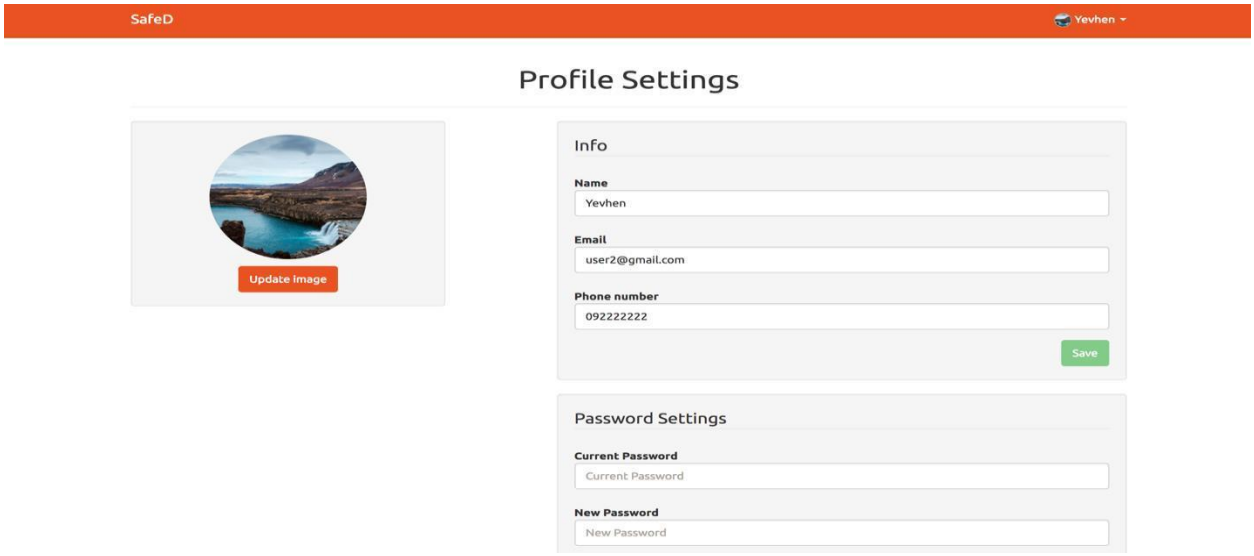

Рисунок 3.3 – Сторінка профайлу

На цій сторінці у користувача є змога змінити свої контактні дані, пароль чи аватар. Також на цій сторінці користувач повинен верифікувати свою карту якщо хоче мати змогу створювати угоди і отримувати гроші. Код цієї сторинки наданий в додатку У. Для цього йому потрібно натиснути кнопку verify після чого на екрані з’явиться модальне вікно де потрібно ввести данні карти (Рисунок 3.4). Лістинг коду модального вікна наданий в додатку Ф.

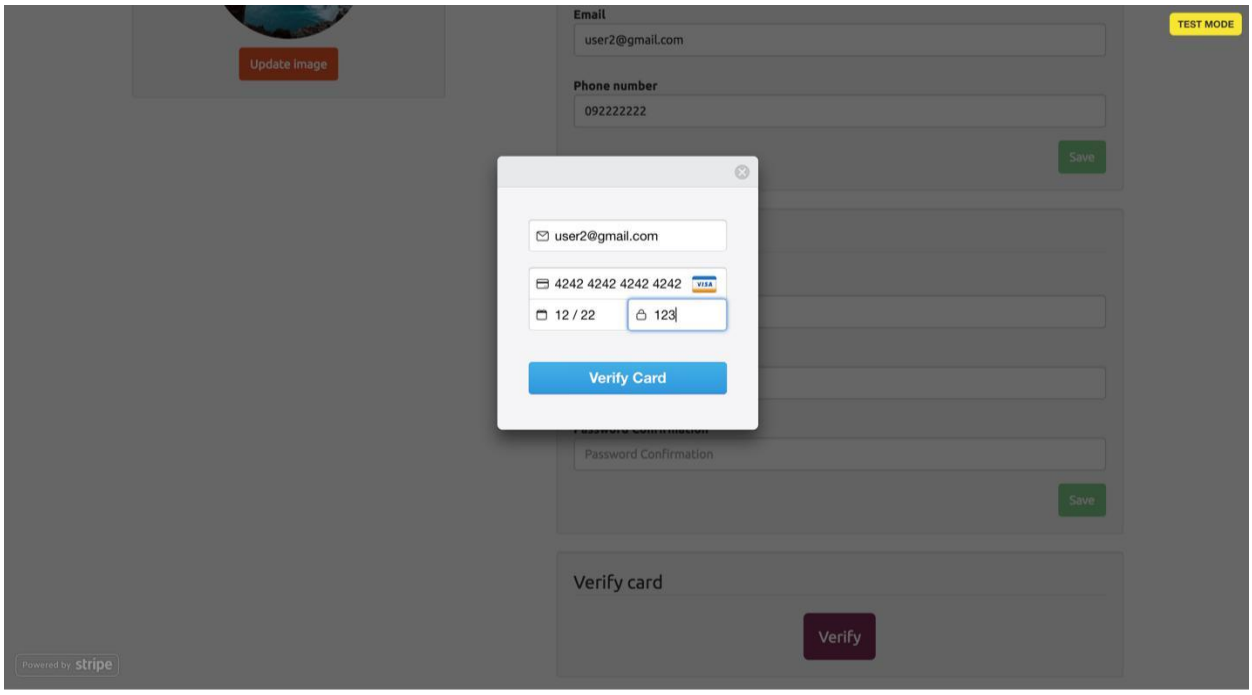

Рисунок 3.4 – Вікно верифікації карти

Після верифікації карти користувач далі може перейти на сторінку Dashboard де є два таби: my deals і deals with me. Також є таблиця зі списком всіх угод користувача (Рисунок 3.5). Лістинг сторінки наведено в додатку Х.

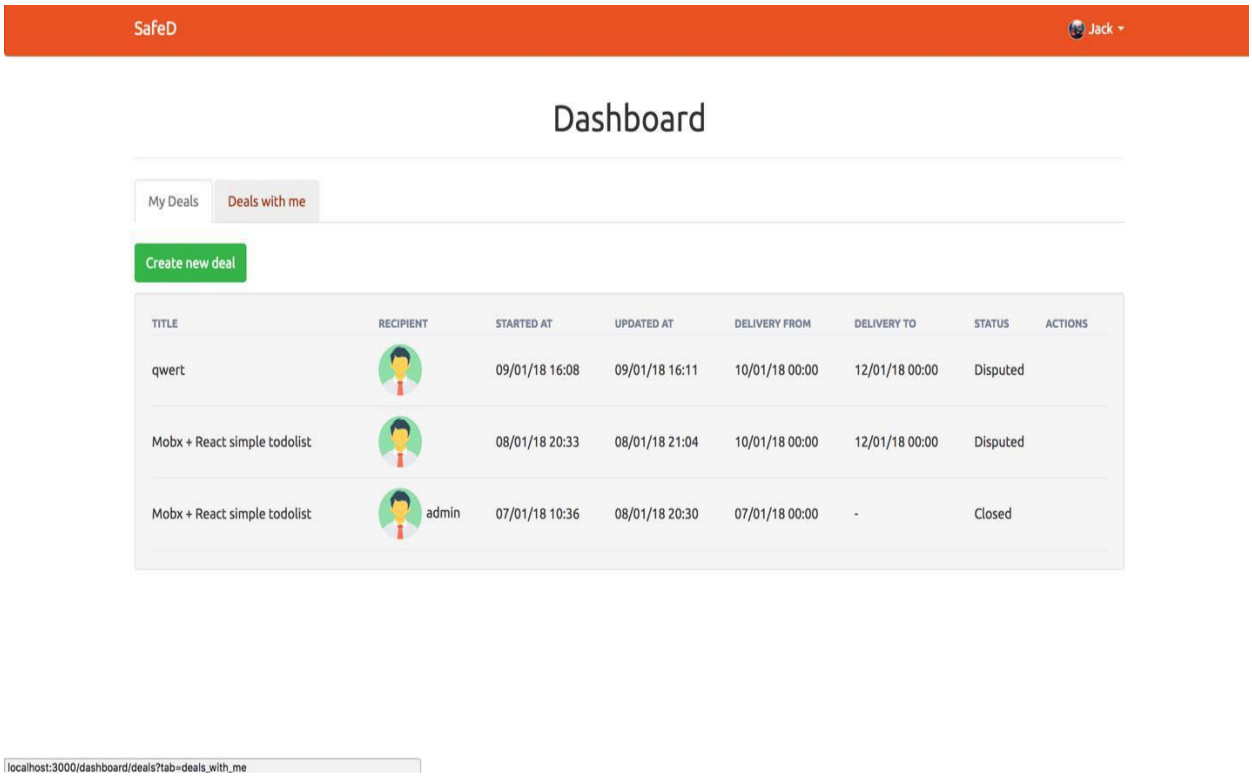

Рисунок 3.5 – Сторінка dashboard

В таблиці вказані назва, отримувач разом з автаром та ім’ям, дата початку, дата зміни статусу, дати доставки і поточний статус угоди і колонка с кнопками які змінюють статус угоди. На вкладці Deals with me поля таблиці трошки інші. Замість поля отримувач з’являється відправник. Також в таблиці на цьому табі.

Також на цій сторінці користувач має змогу створити нову угоду для перерахування грошей за товар. Але модальне вікно може не з’явитись якщо карта ще не була верифікована, натомість ви побачете попередження про те що користувачу потрібно перейти на сторінку профайлу і верифікувати карту.

Для цього йому потрібно натиснути кнопку Create deal після чого на екрані з’явиться модальне вікно с формую для заповнення (Рисунок 3.6).

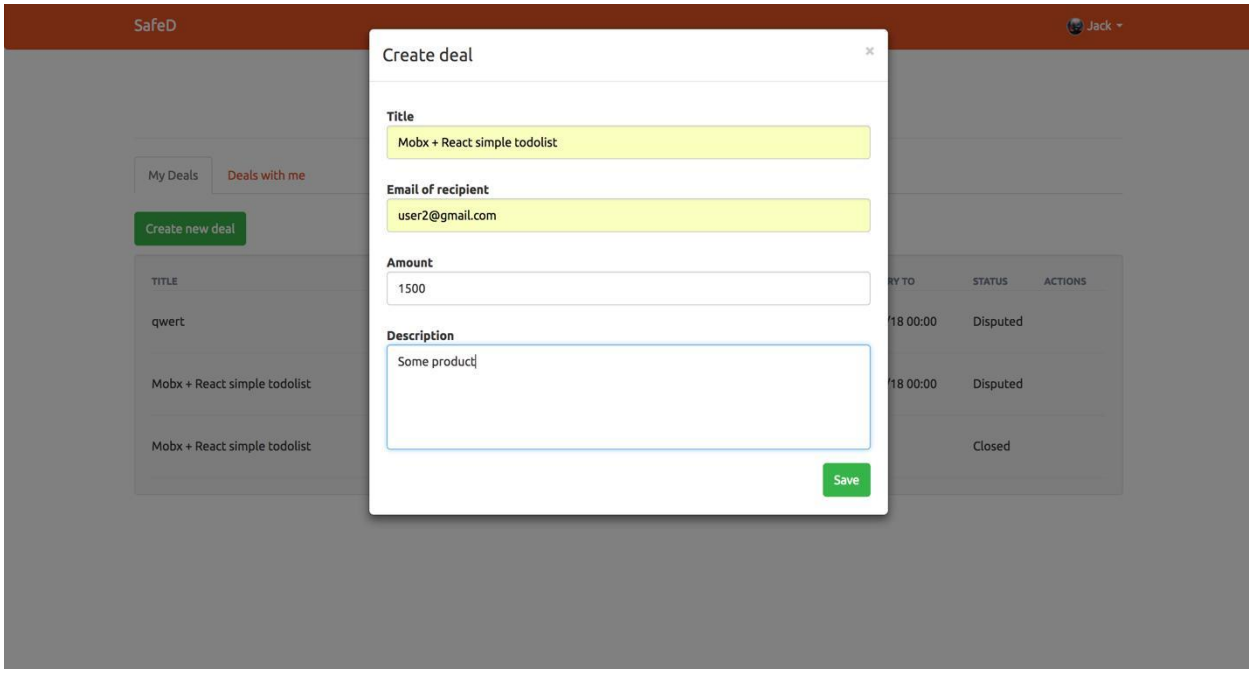

Рисунок 3.6 – Модальне вікно для створення угоди

Щоб створити угоду користувачу потрібно ввести заголовок угоди, поштовий адрес отримувача, суму та опис угоди. Після чого він зможе натиснути Save і угода буде створена а гроші с його карти зняті і будуть залишатись у нас на сервісі поки угода не завершіться, після чого гроші відправляться отримувачу.

Далі користувач побачить створену угоду в таблиці і зможе натиснути кнопку decline яка дасть змогу відмінити угоду, поки отримувач не підтвердне відправку товару. Код форми можна переглянути в додатку Ц, а редюсер угоди де зберігаються дані і описані всі функції оновлення цих даних в додатку Ш.

Отримувач побаче цю угоду на табі deals with me. І в ньго буде можливість відмінити угоду натиснувши decline або відправити товар і натиснути кнопку I send product після чого йому в модальному вікні потрібно буде вибрати період коли повинен прийти товар до користувача який створив угоду (Рисунок 3.7). Тут також спочатку перевіряєтьс чи верифікована карта у користувача, якщо ні тобуде отримано повідомлення про необхідну верифікацію.

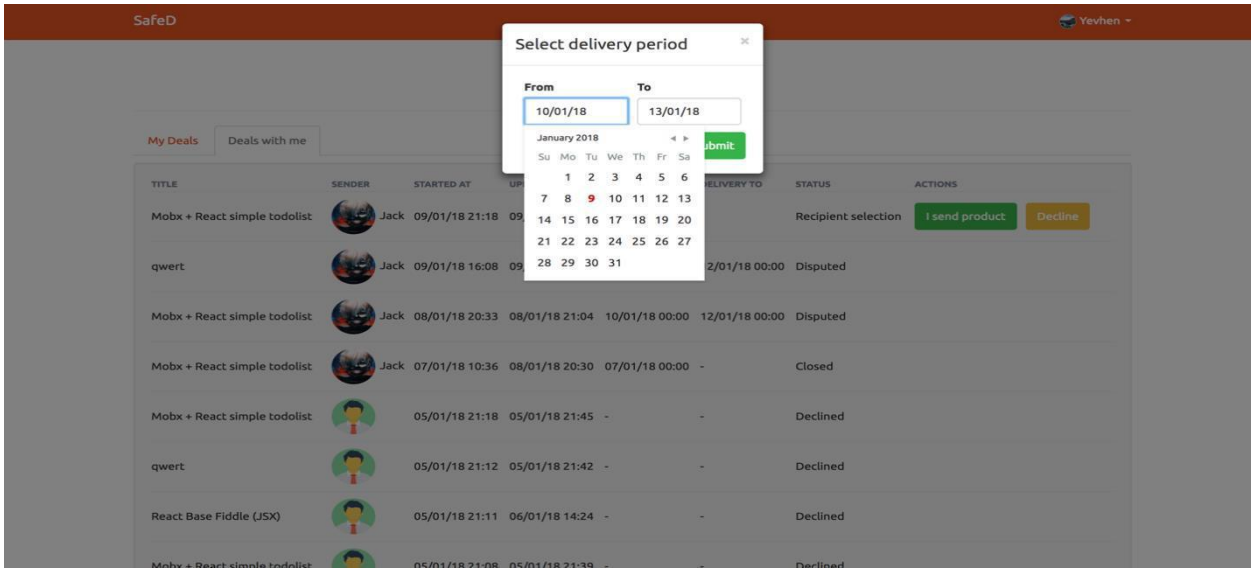

Рисунок 3.7 – Модальне вікно з можливістю вибрати період доставки

Далі у користувачів буде можливість створити суперечку натиснувши кнопку dispute і ввести в поле чим саме вони задоволені. Після чого на поштові скриньки будуть надіслані повідомлення про створення суперечки.

Ще до цієї бесіди буде доданий модератор сервісу, який буде допомагати вирішити суперечку. Вислухавши всі аргументи сторін модератор буде вирішувати кому надіслати гроші. Зробити це у нього буде змога з власного аккаунту. У нього буде схожа таблиця с угодами в стані disputed напроти яких знаходяться дві кнопки For sender і For Recipient (Рисунок 3.8).

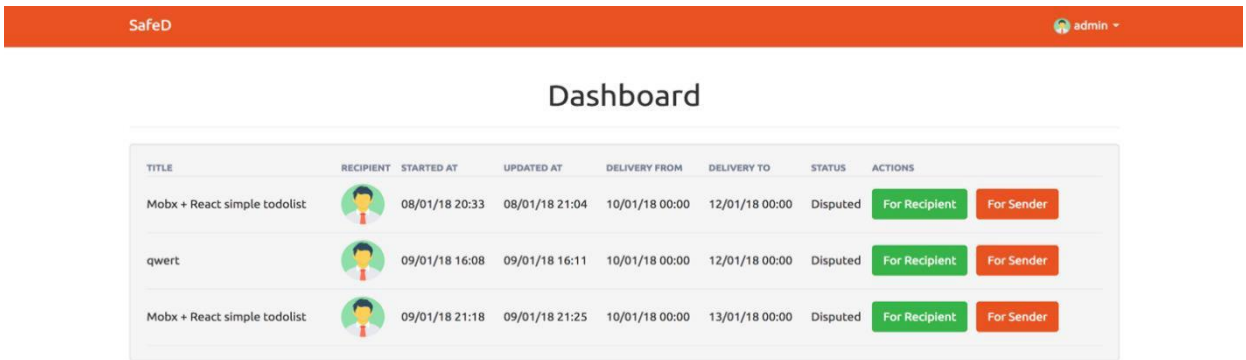

Рисунок 3.8 – Сторінка dashboard для модератору сайту.

Нами був розроблена перша версія нашого веб-додатку. Надалі з кожним новим релізом нових версій ми будемо вдосконалювати наш сервіс і кожний раз впроваджувати нові можливості і покращувати старі.

3.3 Оцінка ефекту від впровадження системи автоматизації

Для розрахунку річного ефекту від провадження нашої системи автоматизації спочатку нами було прораховано витрати зарік на розробку, реліз та рекламу сайту в соціальних мережах.

Витрати:

1. Витрати на розробку (Таблиця 3.1).

Таблиця 3.1 – Витрати на розробку

Витрати	Вартість
Розробка самостійно	100 грн плата за електроенергію та Internet.
Стороння організація	10 доларів * 60 годин розробки дорівнює 600 доларів

2. Витрати на рекламу в соціальних мережах (Таблиця 3.2).

Таблиця 3.2 – Витрати на рекламу

Послуги	Вартість	Результат
Професійне ведення сторінок у Facebook та Вконтакте.	Вартість 3500 грн на місяць	Збільшення кількості фоловеров, коментарів та друзів. Як результат збільшення кількості відвідувачів сайту
Професійне ведення сторінок у Twitter.		Збільшення кількості друзів і ретвітів, разом з цим кількості відвідувачів сайту
Ведення блогів в LiveJournal.		Збільшення кількості друзів та коментарів, разом з цим кількості відидувачів сайту
Соціальні закладки (від 5 до 50 шт.)		При пошуку користувач буде потрапляти на закладки, з якої здійснюється перехід на сайти або профілі в соцмережах.

3. Витрати на реліз. Ціна серверу Simple Cloud де буде знаходитись наш сайт складає 125 грн на місяць. Домене ім’я яке будемо використовувати

280 грн за рік.

Доходи:

1. Дохід від комісії. Ми будемо отримувати з комісії в 0,5 % з суми вказаної в угоді. Для клієнтів які будуть користуватися нашим сервісом частіше, цей процент буде зменшуватися. Припустимо, що 1000 користувачів створять угоди по 100 грн, то це принесе нам відразу 1000 грн.

2. Дохід з поштової розсилки. Зі збільшенням бази клієнтів можна буде почати отримувати прибуток за email розсилку. Приблизно можно отримувати 100 грн за рік з кожного email в базі. Тобто база в 1000 клієнтів мінус 2%. в середньому саме стільки людей відмовляються від розсилки. Розсилка може принести прибуток в розмірі 196000 грн за рік.

Формолізуемо приваедені вище розрахунки за допомогою рисунку 3.1.

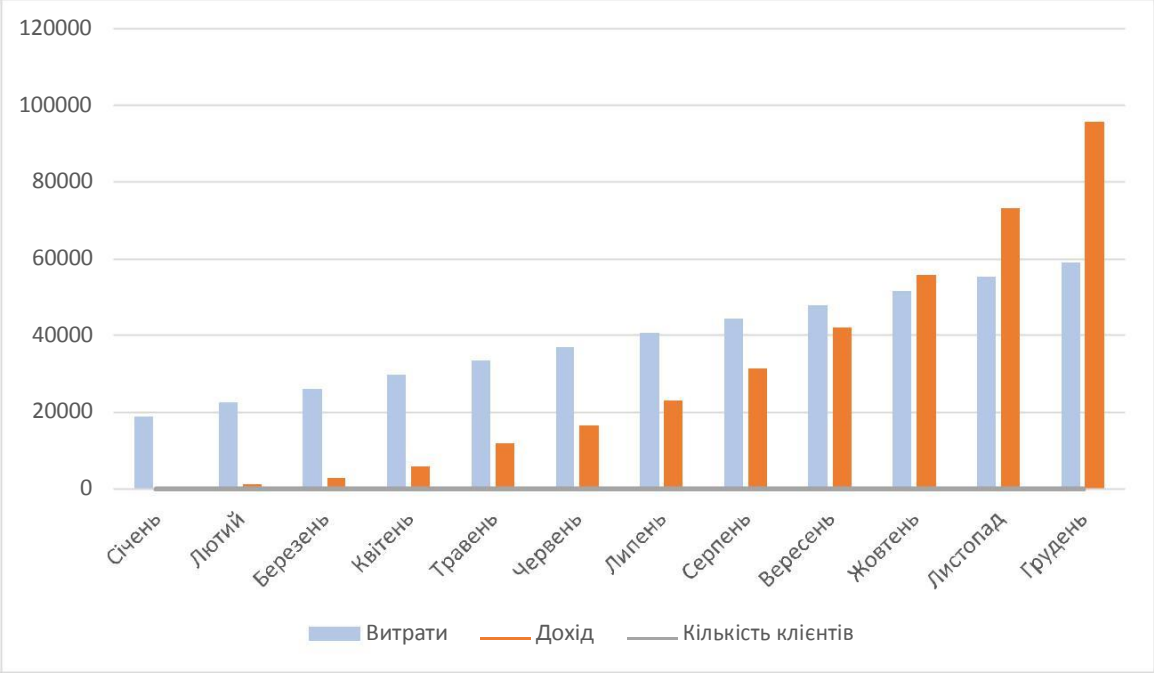

Рисунок 3.1 – Срок окупності розробки

Отже, в нашому випадку срок окупності проекту складу 10 місяців. В серпні ми вже отримаємо наш перший прибуток. С кількістю клієнтів в 815 чоловік з витратами сумою 51530 грн і доходом 55717 грн він складе 4187 грн.

Далі зробимо прогноз на наступний рік і також покажемо його на рисунку 3.2.

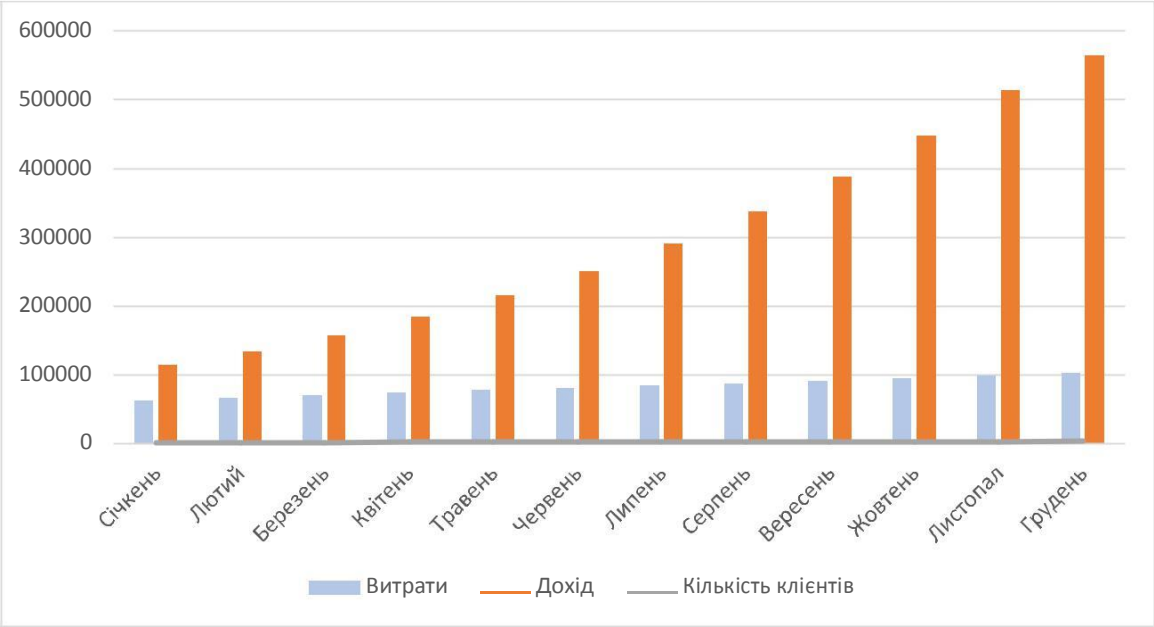

Рисунок 3.2 – Прогноз на наступний рік

Якшо приріст клієнтів і на далі буде складати 10 % то за другий рік наш чистий прибуток буде складати 462499 грн при кількості користувачів 3684 чоловіки.

В таблиці 3.3 наведено статистичні дані за два роки.

Таблиця 3.3 – Статистичні дані за два роки

Рік	Витрати	Дохід	Прибуток	Кількість клієнтів
2018	58780	95574	36794	1174
2019	102280	564779	462499	3685

Отже, як висновок можемо сказати, що наш сервіс є дуже прибутковим. Якщо ввесь час стежити за конкурентами аби потік клієнтів не перетікав до них, а для цього потрібно завжди удосконалювати і робити наш сервіс зручнішим. Також потрібна гарна реклама, що дозволить ввесь час збільшувати кількісь користувачів нашим сервісом. Від яких і залежить наш відсоток доходу.

ВИСНОВОК

У першому пункті першого розділу нами було розглянуто, що собою являють фінансові операції в цілому. Отже, доведено, що під фінансовими операціями слід розуміти угоди та інші дії громодян або юридичних осіб з фінансовими засобами незалежно від форми і способу їх здійснення, пов'язані з переходом права власності та інших прав.

У другому пункті першого розділу були досліджені які є способи та в чому особливості здійснення фінансових операцій в мережі Інтернет. Ми дізнались, що за останні роки розрахунок за товари і послуги в інтернеті набув великої популярності. Дізналися, що таке платіжна система та які системи є найбільш популярні. Тепер можемо сказати, що платежі в інтернеті можуть здійснюватися дотримуючись таких умов: конфіденційність, цілісність, збереження таємниці, автентифікація, авторизація, багатоваріантність засобів оплати, мінімізація плати за транзакцію.

Також розглянувши новий вид електронних грошей, що швидко набирає популярності – криптовалюти, можна вивести найпопулярніші такі як Bitcoin, Ethereum, DASH, Monero и NEM.

У трерьому пункті першого розділу ми розглянули механізми забезпечення безпеки здійснення фінансовий операцій в інтернеті. Отже, найвищій рівень надійності та безпеки забезпечують асиметричні алгоритми шифрування.

У першому пункті другого розділу було сформульовано модель процесу здійснення фінансових операцій в мережи Інтернет та було знайдено вразливі точки. Після чого ми представили свою удосконалену модель здійснення фінансових операцій в інтернеті, з використання сервісу-посередника для підвищення безпеки проведення.

Далі у другому пункті другого розділу було розглянуто переваги і недоліки інформаційних архітектур після чого вибрали архітектуру «кліент - сервер», яка найкраще підходить для вирішення проблеми проведення фінансових операцій в мережі Інтернет. І можемо навести таку її перевагу, як розводілення навантаження між сервером і клієнтською частиною, що дозволить зручно користуватись сервісом одночасно більшій кількості користувачів.

Додатково в третьому пункті другого розділу ми вибрали програмне та технічне забезпечення для розробки нашого автоматизованого вирішення проблеми та описали його переваги.

У першому пункті третього розділу нами було створено прототип нашого інтерфейсу за яким ми пізніше створбвали дизайн нашого сайту. Це було зроблено для того, щоб мати розуміння, як все повинно виглядати і працювати, перш ніж перейти до самої розробки проекту.

Після створення прототипу ми розробили архітектуру бази даних і написталі back-end нашого проекту за допомогою фреймворку Ruby on Rails. Ми вибрали цей фреймворк за його легкість і швидкість розробки.

Далі у пункті два третього розділу ми розробили front-end проекту, після чого було продемонстрованно роботу і надано повну інструкцію з

вікористання веб-додатку для кожного з користувачів нашого сервісу. Ми можемо стверджувати, що інтерфейс є дуже легким і зручним у використані.

У третому пункті третого розділу були розраховані витрати пов’язані з розробкою, рекламою і релізом проєкту. Був підрахований очікуваний ефект по змодельованим даним від впрвадження нашої автоматизованої системи.

Розрахувавши витрати і дохід, можна зробити висновок, що наш сервіс завдяки гарній рекламі в соціальних мережах, може приносити нам велику кількість користувачів, що в свою чергу – чистий прибуток.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Платежі та розрахунки за товари і послуги в електронній комерції. Поняття електронних платіжних систем / [Електронний ресурс] – Режим доступу: http://pidruchniki.com/10590502/informatika/platezhi_rozrahunki_tovari_poslugi_elektronniy_komertsiyi

2. Електронні гроші в Україні: правове підґрунтя та особливості функціонування та Сутність електронних грошей / [Електронний ресурс] – Режим доступу: http://devisu.ua/uk/consultingconsult/6627-elektronni-groshi-v-ukrayini-pravove-pidgruntja-ta-osoblivosti-funktsionuvannja

3. Біткоїн Україна: Майданчик для спілкування та інформаційний центр щодо використання технологій блокчейн (біткоїн) та інших відкритих розподільчих протоколів в Україні / [Електронний ресурс] – Режим доступу:

4. Тенденції розвитку криптовалют на ринку України/ [Електронний ресурс] – Режим доступу: http://www.vestnik-econom.mgu.od.ua/journal/2017/24-1-2017/12.pdf

5. Чайковский Я. І. Платіжні системи / Я. І. Чайковский // Видавництво «Карт-бланш» 2006.

6. Архитектура клієнт-сервер / [Електронний ресурс] – Режим доступу: http://portal.tpu.ru:7777/SHARED/f/FAS/study/avis/lectures/cli-se.pdf

7. Украинцы переходят на онлайн-платежи – НБУ / [Електронний ресурс] – Режим доступу: https://www.segodnya.ua/economics/finance/ukraincy-perehodyat-na-onlayn-platezhi-nbu-701539.html

8. Особливості слідової картини шахрайств, що вчиняються в мережі Інтернет / [Електронний ресурс] – Режим доступу: http://molodyvcheny.in.ua/files/journal/2016/1/57.pdf.

9. Самойлов С. В. Інтернет-аукціони як спосіб шахрайств в мережі

Інтернет / С. В. Самойлов // Співпраця поліції/міліції зі службами безпеки Інтренте-сайтів (аукціонів, соціальних мереж тощо) у боротьбі з інтрент-злочиннічтю на підставі національного законодавства, яке дії у Європейському Союзі : матеріали конф. (м. Хмельницький, 16–17 листопада 2010 р.). – Хмельницький : ГУМВС України в Хмельницькій області, 2010. – С. 75–77.

10. British fraud ran Beijing ticket scam / [Електронний ресурс] – Режим доступу: http://www.smh.com.au/news/off-the-field/olympic-ticket-scammer-unmasked/2008/08/05/1217702042881.html

11. Ticket swindle leaves trail of losers / [Електронний ресурс] – Режим доступу: http://www.smh.com.au/news/latest-news/ticket-swindle-leaves-trail-of-losers/2008/08/03/1217701854125.html

12. Top Online Scams Used by Cyber Criminals to Trick You / [Електронний ресурс] – Режим доступу: https://heimdalsecurity.com/blog/top-online-scams/

13. A Common Currency for Online Fraud; Forgers of U.S. Postal Money Orders Grow in Numbers and Skill / [Електронний ресурс] – Режим доступу: http://query.nytimes.com/gst/fullpage.html?res=9907EED81231F935A15757C0A9639C8B63&n=...&smid=pl-share

14. Теоретичні засади розслідування шахрайства в сучасних умовах. Сутність шахрайства / [Електронний ресурс] – Режим доступу: http://library.nlu.edu.ua/POLN_TEXT/MONOGRAFII_2010/Musienko_2009.pdf

15. Про запобігання та протидію легалізації (відмиванню) доходів, одержаних злочинним шляхом, фінансуванню тероризму та фінансуванню розповсюдження зброї масового знищення» дає специфічне визначення терміну «фінансова операція / [Електронний ресурс] – Режим доступу: http://zakon2.rada.gov.ua/laws/show/1702-18

16. Распределенный вычеслительные системы. Модель «Клиент-Сервер» / [Електронний ресурс] – Режим доступу: http://glebradchenko.susu.ru/doc/Radchenko_Distributed_Computer_Systems.pdf

17. Apple Inc. / [Електронний ресурс] – Режим доступу: https://www.apple.com/

18. What’s new in WebStorm / [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/webstorm/whatsnew/

19. What’s new in RubyMine / [Електронний ресурс] – Режим доступу: https://www.jetbrains.com/ruby/whatsnew/

20. GIT -- distributed-is-the-new-centralized / [Електронний ресурс] – Режим доступу: https://git-scm.com/

21. Web Service Composition: A Survey of Techniques and Tools. ACM Computing Surveys / [Електронний ресурс] – Режим доступу:

22. Achieving Web Services Composition – a Survey / [Електронний ресурс] – Режим доступу: http://article.sapub.org/10.5923.j.se.20120205.03.html

23. Appendix E. Release Notes / [Електронний ресурс] – Режим доступу: https://www.postgresql.org/docs/10/static/release-10-1.html

24. 5 Common Server Setups For Your Web Application / [Електронний ресурс] – Режим доступу: https://www.digitalocean.com/community/tutorials/5-common-server-setups-for-your-web-application

25. MySQL 8.0 Reference Manual / [Електронний ресурс] – Режим доступу: https://dev.mysql.com/doc/refman/8.0/en/

26. Devise_token_auth / [Електронний ресурс] – Режим доступу: https://github.com/lynndylanhurley/devise_token_auth

27. Carrierwave / [Електронний ресурс] – Режим доступу: https://github.com/carrierwaveuploader/carrierwave

28. Minimagic / [Електронний ресурс] – Режим доступу: https://github.com/minimagick/minimagick

29. Active model serializer / [Електронний ресурс] – Режим доступу: https://github.com/rails-api/active_model_serializers

30. AASM / [Електронний ресурс] – Режим доступу: https://github.com/aasm/aasm

31. Free themes for Bootstrap / [Електронний ресурс] – Режим доступу: https://bootswatch.com/

32. React.js / [Електронний ресурс] – Режим доступу: https://reactjs.org/

33. Redux.js / [Електронний ресурс]– Режим доступу: https://redux.js.org/docs/introduction/

34. Про фінансові послуги та державне регулювання ринків фінансових послуг / [Електронний ресурс] – Режим доступу:

35. Про платіжні системи та переказ коштів в Україні / [Електронний ресурс] – Режим доступу: http://zakon2.rada.gov.ua/laws/show/2346-14

36. Про затвердження Положення про електронні гроші в Україні / [Електронний ресурс] – Режим доступу: http://zakon2.rada.gov.ua/laws/show/z0688-08

37. Why Bitcoin is Gaining Traction Україні / [Електронний ресурс] – Режим доступу: https://www.buybitcoinworldwide.com/kb/investing-in-bitcoin

38. Ethereum. Blockchain app platform / [Електронний ресурс] – Режим доступу: https://www.ethereum.org/

39. Dash Cryptocurrency: Everything A Beginner Needs To Know / [Електронний ресурс] – Режим доступу: https://coinsutra.com/dash-cryptocurrency

40. About Monero / [Електронний ресурс] – Режим доступу: https://getmonero.org/resources/about/

41. 10 things you need to know about Ripple / [Електронний ресурс] – Режим доступу: https://www.coindesk.com/10-things-you-need-to-know-about-ripple/

42. Валютні операції та фінансові послуги, що супроводжують їх реалізацію/[Електроннийресурс]–Режимдоступу: http://pidruchniki.com/2015060964924/finansi/valyutni_operatsiyi_finansovi_poslugi_suprovodzhuyut_realizatsiyu

43. ПЕРЕКАЗ КОШТІВ / [Електронний ресурс] – Режим доступу: https://bank.gov.ua/control/uk/publish/article?art_id=123515

44. Спекулятивні операції / [Електронний ресурс] – Режим доступу: https://studme.com.ua/1841041310677/finansy/spekulyativnye_operatsii.htm

45. РАХУНОК ОПЕРАЦІЙ З КАПІТАЛОМ ТА ФІНАНСОВИЙ РАХУНОК ЯК ЗАСІБ ОПТИМІЗАЦІЇ ПЛАТІЖНОГО БАЛАНСУ УКРАЇНИ / [Електронний ресурс] – Режим доступу: http://www.investplan.com.ua/pdf/23_2015/19.pdf

46. Офшорні зони: види та особливості / [Електронний ресурс] – Режим доступу: http://vikna.if.ua/cikavo/68688/view

47. ОФШОРНІ СХЕМИ / [Електронний ресурс] – Режим доступу:

48. Про затвердження Положення про здійснення фінансового моніторингу суб'єктами первинного фінансового моніторингу, державне регулювання та нагляд за діяльністю яких здійснює Міністерство юстиції України / [Електронний ресурс] – Режим доступу: України

49. ЄВРАЗІЙСЬКІ ПЕРСПЕКТИВИ РОЗВИТКУ БАНКІВСЬКИХ СИСТЕМ/[Електроннийресурс]–Режимдоступу: http://banking.uabs.sumdu.edu.ua/images/department/banking/elektro/11.pdf

50. About WebMoney / [Електронний ресурс] – Режим доступу: http://support.worldpay.com/support/kb/gg/alternativepayments/content/webmoney.htm

51. Платежная система Яндекс.Деньги / [Електронний ресурс] – Режим доступу: http://www.banki.ru/wikibank/yandeksdengi/

52. Платежная система RBK Money / [Електронний ресурс] – Режим доступу: http://www.banki.ru/wikibank/rbk_money/

53. At PayPal, we put people at the center of everything we do / [Електронний ресурс] – Режим доступу: https://www.paypal.com/ai/webapps/mpp/ about

54. ПЛАТІЖНІ СИСТЕМИ В ІНТЕРНЕТІ / [Електронний ресурс] – Режим доступу: http://pidruchniki.com/15410104/finansi/platizhni_sistemi_interneti

55. Виконання платежів через Інтернет / [Електронний ресурс] – Режим доступу: http://studopedia.com.ua/1_189694_vikonannya-platezhiv-cherez-Internet.html

56. ОСНОВИ ФУНКЦІОНУВАННЯ ЕЛЕКТРОННИХ ГРОШЕЙ В УКРАЇНІ/[Електроннийресурс]–Режимдоступу: http://ekmair.ukma.edu.ua/bitstream/handle/123456789/2231/Diakovskyi_Osnovy_funktsionuvannia_elektronnykh.pdf

57. Електроні гроші в Україні / [Електронний ресурс] – Режим доступу: http://www.ier.com.ua/files/publications/Books/2012/3_Electronic_Money/E- money_report_APPROVED_2012-10-02_RED2.pdf

58. Електронні гроші в Україні: правове підґрунтя та функціонування / [Електроннийресурс]–Режимдоступу: https://uteka.ua/ua/publication/Elektronnye-dengi-v-Ukraine-pravovaya-osnova-i-funkcionirovanie

59. Что такое криптовалюта? / [Електронний ресурс] – Режим доступу: https://uznayvse.ru/voprosyi/chto-takoe-kriptovaljuta.html

60. Операция "легализация". Каков статус криптовалют в Украине / [Електронний ресурс] – Режим доступу: http://finance.liga.net/cryptoeconomics/ 2017/12/1/articles/55821.html

61. ТРАНСФОРМАЦІЙНІ ПРОЦЕСИ В ЕКОНОМІЦІ УКРАЇНИ: ГЛОБАЛЬНІ ТА РЕГІОНАЛЬНІ АСПЕКТИ / [Електронний ресурс] – Режим доступу: http://ird.gov.ua/irdp/p20170002.pdf

62. ПЕРВЫЙ ДЕНЬ ЗАПУСКА: БИТКОИН ФЬЮЧЕРСЫ НА CME ТОРГУЮТСЯ ПО 20000 ДОЛЛАРОВ / [Електронний ресурс] – Режим доступу: http://newscryptocoin.com/2017/12/18/pervyj-den-zapuska-bitkoin-fyuchersy-na-cme-torguyutsya-po-20000-dollarov

63. РОЗВИТОК КРИПТОВАЛЮТИ НА РИНКУ УКРАЇНИ / [Електронний ресурс] – Режим доступу: http://libfor.com/index.php ?newsid=3091

64. Як купити біткоіни в Україні: що потрібно знати / [Електронний ресурс] – Режим доступу: https://www.obozrevatel.com/ukr/finance/business-and-finance/yak-kupiti-bitkoini-v-ukraini-scho-potribno-znati.htm

65. Криптовалюта - гроші майбутнього / [Електронний ресурс] – Режим доступу: http://vashbankir.com/financial-literacy/articles/kriptovalyuta-dengi-buduschego.html?hl=uk

66. Захист інформаційних систем – важливе завдання сьогодення / [Електронний ресурс] – Режим доступу: http://www.vaas.gov.ua/news/zaxist-informacijnix-sistem-vazhlive-zavdannya-sogodennya/

67. Вимоги до системи захисту інформації / [Електронний ресурс] – Режим доступу: https://studfiles.net/preview/6012701/page:6/

68. Захист інформації при приведенні переказу / [Електронний ресурс] – Режим доступу: http://jurists.org.ua/banking-law/1143-zahist-nformacyi-pri-provedenn-perekazu.html

69. ОСНОВИЗАХИСТУІНФОРМАЦІЇВ ТЕЛЕКОМУНІКАЦІЙНИХ ТА КОМП'ЮТЕРНИХ МЕРЕЖАХ / [Електронний ресурс] – Режим доступу: http://lib.detut.edu.ua/files/Nauk_trud_vukladahiv/Fakultet%20Infrastruktur_ruxomuy_sklad%20%E2%80%9D/Kafedra_tel_texn_avtomatuka/nauk_trud_voznenko.pdf

70. АСИМЕТРИЧНІ МЕТОДИ ШИФРУВАННЯ В ТЕЛЕКОМУНІКАЦІЯХ / [Електронний ресурс] – Режим доступу:

71. Електронна комерція з використанням засобів Інтернету / [Електронний ресурс] – Режим доступу: http://westudents.com.ua/glavy/100 221-53-elektronna-komertsya-z-vikoristannya m-zasobv-nternetu.html

72. Secret key algorithm (symmetric algorithm) / [Електронний ресурс] – Режим доступу: http://searchsecurity.techtarget.com/definition/secret-key-algorithm

73. Asymmetric algorithms / [Електронний ресурс] – Режим доступу: https://cryptography.io/en/latest/hazmat/primitives/asymmetric/

74. Загальна концепція платіжної системи / [Електронний ресурс] – Режим доступу: http://bookss.co.ua/book_zagalna-koncepciya-platizhno-sistemi_911/5_tema-4.-bezpeka-platizhnih-sistem

75. «Ви — спадкоємець мільйонних статків родича з Африки» / [Електронний ресурс] – Режим доступу: http://www.beregszasz.com.ua /index.php/cikkek-br-statti/5692-vy-spadkoiemets-milionnykh-statkiv-rodycha-z-afryky

76. Що таке фішинг? / [Електронний ресурс] – Режим доступу: https://ua.godaddy.com/help/sho-take-fishing-346

ДОДАТКИ

ДОДАТОК А

SUMMARY

Cheshchevyi Y. I. Automation of improving security of financial transactions in Internet. – Masters-level Qualification Thesis. Sumy State University, Sumy, 2018.

The master's thesis focuses on the process of conducting financial transactions on the Internet. The analysis of information systems and technologies for financial transactions on the Internet is carried out. The main purpose of this research is to develop an automated solution to the problem of security of financial transactions in the Internet

Key words: financial operations, Internet security, information technologies, fraud in the Interne, cybercrime.

АНОТАЦІЯ

Чещевий Є. І. Автоматизація процесу підвищення безпеки проведення фінансових операцій в мережі Internet. – Кваліфікаційна магістерська робота. Сумський державний університет, Суми, 2018 р.

У роботі було досліджено процес здійснення фінансових операцій в мережі Internet. Проведений аналіз інформаційних систем та технологій здійснення фінансових операцій в інтернеті.

Основною метою цього дослідження є розробка автоматизованого вирішення проблеми безпеки здійснення фінансових операцій в мережі Internet

Ключові слова: фінансові операції, безпека в інтернеті, інформаційні технології, інтернет-шахрайство, кібершахрайство.

ДОДАТОК Б

Міграції створення користувача

```
class DeviseTokenAuthCreateUsers < ActiveRecord::Migration[5.1]
  def change
    create_table(:users) do |t|
      ## Required
      t.string :provider, :null => false, :default => "email"
      t.string :uid, :null => false, :default => ""
      ## Database authenticatable
      t.string :encrypted_password, :null => false, :default => ""
      ## Recoverable
      t.string   :reset_password_token
      t.datetime :reset_password_sent_at
      ## Rememberable
      t.datetime :remember_created_at
      ## Trackable
      t.integer  :sign_in_count, :default => 0, :null => false
      t.datetime :current_sign_in_at
      t.datetime :last_sign_in_at
      t.string   :current_sign_in_ip
      t.string   :last_sign_in_ip
      ## Confirmable
      t.string   :confirmation_token
      t.datetime :confirmed_at
      t.datetime :confirmation_sent_at
      t.string   :unconfirmed_email # Only if using reconfirmable
      ## Lockable
      # t.integer  :failed_attempts, :default => 0, :null => false # Only if lock strategy is :failed_attempts
      # t.string   :unlock_token # Only if unlock strategy is :email or :both
      # t.datetime :locked_at
      ## User Info
      t.string :name
      t.string :nickname
      t.string :image
      t.string :email
      ## Tokens
      t.json :tokens
      t.timestamps
    end
    add_index :users, :email,                unique: true
    add_index :users, [:uid, :provider],     unique: true
    add_index :users, :reset_password_token, unique: true
    add_index :users, :confirmation_token,   unique: true
    # add_index :users, :unlock_token,       unique: true
  end
end
class AddImageUrlsToUser < ActiveRecord::Migration[5.1]
  def change
    add_column :users, :avatar, :string
  end
end
class AddRoleToUser < ActiveRecord::Migration[5.1]
  def change
    add_column :users, :role, :string, default: 'user'
  end
end
```

ДОДАТОК В

Файл routes.rb

```
Rails.application.routes.draw do
  scope :api do
    mount_devise_token_auth_for 'User', at: 'auth', skip: [:omniauth_callbacks], controllers:
      {
        registrations: 'auth/registrations',
        sessions: 'auth/sessions',
        token_validations: 'auth/token_validations',
        passwords: 'auth/passwords'
      }

    resources :users, only: [:show] do
      put :verify, on: :member
    end
    resources :deals, only: %i(index create update) do
      put :decline, on: :member
      put :send_product, on: :member
      put :start_dispute, on: :member
      put :close, on: :member
    end
  end
end
```

ДОДАТОК Г

Кастомні контролери для devise_token_auth

```
class Auth::PasswordsController <
  DeviseTokenAuth::PasswordsController def render_create_success
    render_resource_data(@resource)
  end
end
class Auth::RegistrationsController <
  DeviseTokenAuth::RegistrationsController def render_create_success
    render_resource_data(@resource)
  end
  def render_create_error
    render_resource_errors(@resource)
  end
  def render_update_success
    render_resource_data(@resource)
  end
  def render_update_error
    render_resource_errors(@resource)
  end
end
class Auth::TokenValidationsController <
  DeviseTokenAuth::TokenValidationsController def render_validate_token_success
    render_resource_data(@resource)
  end
end
```

ДОДАТОК Д

Файл base_controller_methods.rb

```
module BaseControllerMethods
  extend ActiveSupport::Concern
  def render_resource_or_errors(resource, options = {})
    if resource.errors.empty?
      render_resource_data(resource, options)
    else
      render_resource_errors(resource)
    end
  end
  def render_resource_data(resource, options = {})
    render options.merge(json: resource, root: :resource)
  end
  def render_resource_errors(resource)
    render json: { errors: resource.errors }, status: :unprocessable_entity
  end
  def render_resources(resources, options = {})
    options[:pagination] = true if options[:pagination].nil?
    pagination = options.delete(:pagination)
    resources = resources.to_a.uniq
    total = resources.respond_to?(:total_count) ? resources.total_count : resources.length
    default = { root: :resources, meta: { total: total } }
    results = pagination ? Kaminari.paginate_array(resources).page(params[:page]) .per(params[
:per_page]) : resources
    render({ json: results }.merge(default).merge(options))
  endend
```

ДОДАТОК Е

Контролер api_controller.rb

```
class ApiController < ActionController::API
  include DeviseTokenAuth::Concerns::SetUserByToken
  include ActionController::Helpers
  include BaseControllerMethods

  include Pundit

  after_action :verify_authorized, unless: :devise_controller?

  def permitted_attributes(record)
    params.require(:resource).permit(policy(record).permitted_attributes)
  end

  private

      rescue_from ActionController::ParameterMissing do |exception|
        render json: { errors: exception.message }, status: :bad_request
  end
  rescue_from ActiveRecord::RecordNotFound do |exception|
    render json: { errors: exception.message }, status: :not_found
  end
  rescue_from Pundit::NotAuthorizedError do
    render json: { errors: 'You are not authorized to access this action.' }, status:
  :forbidden end
end
```

ДОДАТОК Ж

Файл application_controller.rb

```
class ApplicationController < ActionController::Base
  include ActionController::Serialization
  include BaseControllerMethods

  before_action :configure_permitted_parameters, if: :devise_controller?

  protected

  def configure_permitted_parameters
    devise_parameter_sanitizer
      .permit(:sign_up, keys: %i[email password password_confirmation])

    devise_parameter_sanitizer
      .permit(:account_update,
              keys: UserPolicy.new(nil, nil).permitted_attributes)
  end
end
```

ДОДАТОК З

Модель user.rb

```
class User < ActiveRecord::Base
  devise :database_authenticatable, :registerable, :recoverable, :rememberable, :trackable, :validatable

  include DeviseTokenAuth::Concerns::User
  mount_uploader :avatar, AvatarUploader
  has_many :deals, -> { order('created_at desc') }
  has_many :deals_with_me,  -> { order('created_at desc') }, class_name: 'Deal',  foreign_key:
'recipient_id'
  def verify_card params
    if params['stripe_card_id']
      update(verified: true)
    else
      errors.add(:base, 'No stripe card id found')
    end
    self
  end
end
```

ДОТАТОК И

Список гемів. Файл Gemfile

```
# frozen_string_literal: true

source 'https://rubygems.org'

git_source(:github) do |repo_name|
  repo_name = "#{repo_name}/#{repo_name}" unless repo_name.include?("/")
  "https://github.com/#{repo_name}.git"
end

gem 'aasm'
gem 'active_model_serializers', '~> 0.10.0'
gem 'annotate'
gem 'carrierwave', '~> 1.0'
gem 'carrierwave-data-uri'
gem 'devise'
gem 'devise_token_auth'
gem 'kaminari'
gem 'mini_magick'
gem 'pg', '~> 0.21.0'
gem 'puma', '~> 3.7'
gem 'pundit'
gem 'rails', '~> 5.1.4'
group :development do
  gem 'spring'
  gem 'spring-watcher-listen', '~> 2.0.0'
end
```

ДОТАТОК К

Модел deal.rb

```
class Deal < ApplicationRecord
  belongs_to :user
  has_one :recipient, class_name: 'User', foreign_key: 'id'

  include AASM

  enum state: {
    recipient_selection: 0,
    pending_of_obtaining: 1,
    closed: 2,
    disputed: 3,
    declined: 4
  }

  aasm column: :state, enum: true do
    state :recipient_selection, initial: true
    state :pending_of_obtaining
    state :closed
    state :disputed
    state :declined

    event :decline do
      transitions from: [:recipient_selection, :disputed], to:
    :declined end

    event :send_product do
      transitions from: :recipient_selection, to:
    :pending_of_obtaining end

    event :start_dispute, after: :send_dispute_email do
      transitions from: :pending_of_obtaining, to: :disputed
    end

    event :close do
      transitions from: [:pending_of_obtaining, :disputed], to:
    :closed end

  end

  private

  def send_dispute_email
    recipient_user = recipient.present? ? recipient : User.find(recipient_id) recipients
    = [user.email, recipient_user.email, 'moderator@gmail.com'] recipients.each{ |n|
    DisputeMailer.dispute_email(self, n, recipient_user).deliver }
  end

end
```

ДОДАТОК Л

Файл production.rb

```
Rails.application.configure do
  config.cache_classes = true
  config.eager_load = true
  config.consider_all_requests_local       = false
  config.action_controller.perform_caching = true
  config.read_encrypted_secrets = true
  config.public_file_server.enabled = ENV['RAILS_SERVE_STATIC_FILES'].present?
  config.assets.js_compressor = :uglifier
  config.assets.compile = false
  config.log_level = :debug
  config.log_tags = [ :request_id ]
  config.action_mailer.perform_caching = false
  config.i18n.fallbacks = true
  config.active_support.deprecation = :notify
  config.log_formatter = ::Logger::Formatter.new
  if ENV["RAILS_LOG_TO_STDOUT"].present?
    logger           = ActiveSupport::Logger.new(STDOUT)
    logger.formatter = config.log_formatter
    config.logger    = ActiveSupport::TaggedLogging.new(logger)
  end
  config.active_record.dump_schema_after_migration = false
  config.action_mailer.delivery_method = :smtp
  config.action_mailer.smtp_settings = {
    :address              => "smtp.gmail.com",
    :port                 => 587,
    :user_name            => ENV['gmail_username'],
    :password             => ENV['gmail_password'],
    :authentication       => "plain",
    :enable_starttls_auto => true
  }
end
```

ДОДАТОК М

Файл deals_controller.rb

```
class DealsController < ApiController

  def create
    authorize Deal
    user = User.find_by_email(params[:resource][:email_of_recipient])
    if user.present?
      deal = current_user.deals.create permitted_attributes(Deal).merge(recipient_id: user.id)
      render_resource_or_errors(deal)
    else
      render json: { errors: 'This user is not found' }, status:
    :unprocessable_entity end
  end

  def index
    authorize Deal
    render_resources current_user.role == 'admin' ? Deal.where(state: 3) : user_deal,
each_serializer: deal_serializer
  end

  def decline
    authorize Deal
    deal = Deal.find(params[:id])
    deal.decline!
    render_resource_or_errors(deal)
  end

  def send_product
    deal = Deal.find(params[:id])
    authorize deal
    deal.update send_product_params
    deal.send_product!
    render_resource_or_errors(deal)
  end

  def start_dispute
    deal = Deal.find(params[:id])
    authorize deal
    deal.update dispute_params
    deal.start_dispute!
    render_resource_or_errors(deal)
  end

  def close
    deal = Deal.find(params[:id])
    authorize deal
    deal.close!
    render_resource_or_errors(deal)
```

```
  end

  private
  def user_deal
    params[:deals_type] == 'my_deals' ? current_user.deals :
  current_user.deals_with_me end

  def deal_serializer
    params[:deals_type] == 'my_deals' ? SenderDealSerializer :
  DealSerializer end

  def send_product_params
    params.require(:resource).permit(:delivery_to, :delivery_from)
  end

  def dispute_params
    params.require(:resource).permit(:message)
  end
end
```

ДОДАТОК Н

Файли deal_serializer.rb і sender_deal_serializer.rb

```
class DealSerializer < ActiveModel::Serializer
  attributes :id, :user, :state, :title, :amount, :description, :created_at, :updated_at, :delivery_to, :delivery_from

end
class SenderDealSerializer < DealSerializer
  attributes :recipient
end
```

ДОДАТОК П

Сторінка реєстрації

1) Компонент Register

```
import React from 'react'
import PropTypes from 'prop-types'
// components
import { Field, SubmissionError } from 'redux-form'
import Input from 'components/FormFields/Input'
import { toastr } from 'react-redux-toastr'
// styles
import classNames from 'classnames/bind'
import styles from './styles.scss'
const cx = classNames.bind(styles)
export default class Register extends React.Component {
  static propTypes = {
    handleSubmit: PropTypes.func.isRequired,
    pristine: PropTypes.bool.isRequired,
    submitting: PropTypes.bool.isRequired,
    registration: PropTypes.func.isRequired
  }

  onSubmit = (data) => {
    return this.props.registration(data)
      .then(() => toastr.success('Success!', 'You successfully registered!'))
      .catch(response => {
        throw new SubmissionError(response.errors)
      })
  }

  render () {
    const { handleSubmit, pristine, submitting } = this.props
    return (
      <div>
        <h1 className='text-center'>Safe Deal</h1>
        <div className={cx('panel panel-primary', 'panel-
          wrapper')}> <div className='panel-heading'>
            <h2 className='panel-title'>Join to Safe
          Deal</h2> </div>
          <div className='panel-body'>
            <form className='form-horizontal left-aligned'
onSubmit={handleSubmit(this.onSubmit)}>
              <div className='form-group'>
                <div className='col-lg-12'>
                  <Field
                    className='form-control'
                    name='email'
```

```
                type='email'
                placeholder='Email'
                component={Input}
              />
            </div>
          </div>
          <div className='form-group'>
            <div className='col-lg-12'>
              <Field
                className='form-control'
                name='password'
                type='password'
                placeholder='Password'
                component={Input}
              />
            </div>
          </div>
          <div className='form-group'>
            <div className='col-lg-12'>
              <Field
                className='form-control'
                name='password_confirmation'
                type='password'
                placeholder='Confirmation password'
                component={Input}
              />
            </div>
          </div>
          <div className='text-right'>
            <button type='submit' className='btn btn-success' disabled={pristine ||
              submitting}> Register
            </button>
          </div>
        </form>
      </div>
    </div>
  </div>
 )
 }
}
```

2) Контейнер RegisterContainer

```
import { connect } from 'react-redux'
import Register from '../components/Register'
import { reduxForm } from 'redux-form'
import { registration } from '../../../store/modules/auth'
const mapActionCreators = {
  registration}export default connect(null, mapActionCreators)(reduxForm({ form: 'register-
form' })(Register))
```

ДОДАТОК Р

Файл index.js в папці routes

```
import CoreLayout from '../layouts/PageLayout/PageLayout'
import { loadCurrentUser } from '../store/modules/auth' export
const createRoutes = (store, authData) => {
  function preloadUser (nextState, replace, proceed) {
    const { user } = store.getState().auth
    if (!user) {
      if (authData.uid) {
        store.dispatch(loadCurrentUser())
          .then(() => proceed())
          .catch(() => proceed())
        return
      }
    }
    proceed()
  }
  const router = {
    path: '/',
    component: CoreLayout,
    indexRoute: { onEnter: (nextState, replace) => replace('/dashboard') },
    onEnter: preloadUser,
    client: store.client,
    getChildRoutes (location, cb) {
      require.ensure([], (require) => {
        cb(null, [
          require('./Register').default,
          require('./Login').default,
          require('./Profile').default,
          require('./Dashboard').default(store)
        ])
      })
    }
  }
  return router
}
export default createRoutes
```

ДОДАТОК С
Файл auth.js

```
import { clearData } from '../helpers/authData'
import { push } from 'react-router-redux'
// ------------------------------------
// Constants
// ------------------------------------
const SET_USER = 'auth/SET_USER'
const LOGOUT = 'auth/LOGOUT'
const LOADING = 'auth/LOADING'
// ------------------------------------
// Actions
// ------------------------------------
export const registration = (data) => {
  return (dispatch, getState, client) =>
    client.post('auth', { data, auth: true })
      .then((response) => {
        dispatch({ type: SET_USER, payload: response.resource
      }) })
}
export const login = (data) => {
  return (dispatch, getState, client) =>
    client.post('auth/sign_in', { data, auth: true })
      .then((response) => {
        dispatch({ type: SET_USER, payload: response.resource
      }) })
}
export const loadCurrentUser = () => {
  return (dispatch, getState, client) =>
    client.get('auth/validate_token', { auth: true })
      .then((response) => {
        dispatch({ type: SET_USER, payload: response.resource
      }) })
      .catch(() => {
        console.log('redirect')
        dispatch(push('/login'))
      })
}
export const logout = () => { return
  (dispatch, getState, client) =>
    client.del('auth/sign_out')
      .then(() => {
        dispatch({ type: LOGOUT, payload: null
        }) clearData()
      })
}
export const updateUser = (data) => {
  return (dispatch, getState, client) => {
    dispatch(setLoading(true))
```

```
    return client.put('/auth', { data })
      .then((response) => {
        dispatch({ type: SET_USER, payload: response.resource })
        dispatch(setLoading(false))
      })
  }
}
export const setUser = (user) => {
  return (dispatch) => {
    dispatch({ type: SET_USER, payload: user })
  }
}
const setLoading = (bool) => ({ type: LOADING, payload: bool })
// ------------------------------------
// Action Handlers
// ------------------------------------
const ACTION_HANDLERS = {
  [SET_USER]: (state, action) => ({ ...state, user: action.payload }),
  [LOGOUT]: (state, action) => ({ ...state, user: action.payload }),
  [LOADING]: (state, action) => ({ ...state, loading: action.payload })
}
// ------------------------------------
// Reducer
// ------------------------------------
const initialState = {
  user: null, loading:
  false
}
export default function counterReducer (state = initialState, action)
  { const handler = ACTION_HANDLERS[action.type]
  return handler ? handler(state, action) : state
}
```

ДОДАТОК Т

Шапка сайту

1) Header.js

```
import React from 'react'
import PropTypes from 'prop-types'
// components
import { NavDropdown, MenuItem, Navbar, Nav, Button } from 'react-bootstrap'
import { IndexLink } from 'react-router'
import { LinkContainer } from 'react-router-bootstrap'
import classNames from 'classnames/bind'
// utils
import get from 'lodash/get'
// styles
import styles from './styles.scss'
import avatarPlaceholder from 'static/man.svg'
const cx = classNames.bind(styles)
export default class Header extends React.Component {
  static propTypes = {
    logout: PropTypes.func.isRequired,
    user: PropTypes.object
  }
  static contextTypes = {
    router: PropTypes.object.isRequired
  }
  logout = () => {
    this.props.logout()
      .then(() => {
        this.context.router.push('/login')
      })
  }
  redirectTo = (route) => () => {
    this.context.router.push(`/${route}`)
  }
  render () {
    const { user } = this.props
    const userDropdownTitle = (user &&
      <span className={cx('user-dropdown-box')}>
        <img className={cx('user-avatar')} src={get(user, 'avatar.small.url') || avatarPlaceholder}
/>
         
        {user.name || user.email}
      </span>
    )
    const userDropdown = (user &&
      <Nav pullRight>
        <NavDropdown id='nav-dropdown'
          title={userDropdownTitle}> <LinkContainer to='/profile'>
            <MenuItem>Profile</MenuItem>
```

```
        </LinkContainer>
        <LinkContainer to='/dashboard'>
          <MenuItem>Dashboard</MenuItem>
        </LinkContainer>
        <MenuItem divider />
        <MenuItem onClick={this.logout}>Logout</MenuItem>
      </NavDropdown>
    </Nav>
  )
  return (
    <Navbar className='navbar navbar-default'>
      <Navbar.Header>
        <Navbar.Brand>
          <IndexLink to='/' className='navbar-
        brand'>SafeD</IndexLink> </Navbar.Brand>
      </Navbar.Header>
      {user
        ? userDropdown
        : <Navbar.Form pullRight>
          <Button className='btn btn-success'
          onClick={this.redirectTo('login')}>Login</Button>    
          <Button className='btn btn-warning'
onClick={this.redirectTo('register')}>Register</Button>
        </Navbar.Form>
      }
    </Navbar>
  )
 }
}
```

2) HeaderContainer.js

```
import { connect } from 'react-redux'
import Header from '../../components/Header'
import { logout } from '../../store/modules/auth'
const mapActionCreators = {
 logout
}
const mapStateToProps = (state) => ({
 user: state.auth.user
})
export default connect(mapStateToProps, mapActionCreators)(Header)
```

3) styles.scss

```
@import '../../styles/variables';
.user-dropdown-box {
 display: inline-block;
 max-width: 250px;
 overflow: hidden;
 white-space: nowrap;
 text-overflow: ellipsis;
 vertical-align: middle;}
```

ДОДАТОК У

Сторінка профайлу

1) Компонент Profile

```
import React from 'react'
import PropTypes from 'prop-types'
// components
import ProfileForm from './ProfileForm'
import PasswordForm from './PasswordForm'
import VerifyCard from './VerifyCard'
import ImageUploader from '../../../components/ImageUploader'
import { toastr } from 'react-redux-toastr'
import { SubmissionError } from 'redux-form'
import Spinner from '../../../components/Spinner'
// utils
import pick from 'lodash/pick'
import get from 'lodash/get'
import omit from 'lodash/omit'
export default class Profile extends React.Component
  { static propTypes = {
    updateUser: PropTypes.func.isRequired,
    showModal: PropTypes.func.isRequired,
    resetForm: PropTypes.func.isRequired,
    loadCurrentUser: PropTypes.func.isRequired,
    loading: PropTypes.bool.isRequired,
    user: PropTypes.object
  }

  componentDidMount () {
    if (!this.props.user) this.props.loadCurrentUser()
  }

  handleSubmit = (data) => {
    return this.props.updateUser(omit(data, ['avatar']))
      .then(() => {
        toastr.success('Success!', 'User info has been updated!')
      })
      .catch(({ errors }) => {
        throw new SubmissionError(errors)
      })
  }

  handlePasswordSubmit = (data) => {
    if (!data.current_password) {
      throw new SubmissionError({ current_password: ['can\'t be blank'] })
    }
    return this.props.updateUser(data)
      .then(() => {
        this.props.resetForm('password-form')
```

```
      toastr.success('Success!', 'Password has been changed!')
    })
    .catch(({ errors }) => {
      throw new SubmissionError(errors)
    })
  }

  render () {
    const { user, showModal, loading } = this.props
    const initialValues = pick(user, ['name', 'email', 'phone', 'avatar'])

    return (
      <div>
        <h1 className='text-center'>Profile
        Settings</h1> <hr />
        <div className='row'>
          <div className='col-sm-4'>
            <ImageUploader
              url={get(initialValues, 'avatar.medium.url')}
              showModal={showModal}
              updateImage={this.handleSubmit}
            />
          </div>
          <div className='col-sm-7 col-sm-offset-1'>
            <ProfileForm initialValues={initialValues} onSubmit={this.handleSubmit}
            /> <PasswordForm onSubmit={this.handlePasswordSubmit} />
            <VerifyCard />
          </div>
        </div>
        {loading && <Spinner />}
      </div>
    )
  }
}
```

2) Контеінер ProfileContainer

```
import { connect } from 'react-redux'
import Profile from '../components/Profile'
import { reset as resetForm } from 'redux-form'
import { updateUser, loadCurrentUser } from '../../../store/modules/auth'
import { showModal } from '../../../store/modules/modals'
const mapActionCreators = {
  updateUser,
  showModal,
  loadCurrentUser,
  resetForm
}
const mapStateToProps = (state) => ({
  user: state.auth.user,
  loading: state.auth.loading
})export default connect(mapStateToProps, mapActionCreators)(Profile)
```

ДОДАТОК Ф

Код модального вікна верифікації карти

1) Компонент VerefiedCard

```
import React, { Component } from 'react'
import PropTypes from 'prop-types'
// components
import StripeCheckout from 'react-stripe-checkout'
import { toastr } from 'react-redux-toastr'
// utils
import handleResponseErrors from 'utils/handleResponseErrors'
export default class VerifiedCard extends Component {
  static propTypes = {
    user: PropTypes.object.isRequired,
    setUser: PropTypes.func.isRequired
  }
  static contextTypes = {
    client: PropTypes.object.isRequired,
    router: PropTypes.object.isRequired
  }
  verifyCard = (data) => {
    const { user, setUser } = this.props
    this.context.client.put(`/users/${user.id}/verify`, {
      data: {
        resource: {
          stripe_card_id: data.id
        }
      }
    }).then((response) => {
      setUser(response.resource)
      toastr.success('Success', 'You have successfully verified your card')
    }).catch(e => handleResponseErrors(e, ['base']))
  }

  render () {
    return (
      <div className='well'>
        <legend className='text-left'>Verify
        card</legend> <StripeCheckout
          token={this.verifyCard}
          stripeKey={__STRIPE_KEY__}
          panelLabel='Verify Card'
          allowRememberMe={false}
        >
          <div className='text-center'>
            <button type='button' className='btn btn-info btn-
              lg'> Verify
            </button>
          </div>
```

```
      </StripeCheckout>
    </div>
  )
 }
}
```

2) Контеінер VerifiedCardContainer

```
import { connect } from 'react-redux'
import VerifiedCard from './VerifiedCard'
import { setUser } from 'store/modules/auth'

const mapActionCreators = {
 setUser
}

const mapStateToProps = (state) => ({
 user: state.auth.user
})

export default connect(mapStateToProps, mapActionCreators)(VerifiedCard)
```

ДОДАТОК Х

Код сторінки dashboard

1) Компонент Deals

```
import React, { Component } from 'react'
import PropTypes from 'prop-types'
// components
import CreateDealModal from '../../modals/CreateDeal'
import ChooseDeliveryPeriodModal from '../../modals/ChooseDeliveryPeriod'
import DealTable from 'components/DealTable'
import classNames from 'classnames/bind'
import { LinkContainer } from 'react-router-bootstrap'
import { NavItem, Nav, OverlayTrigger, Popover } from 'react-bootstrap'
import { toastr } from 'react-redux-toastr'
// styles
import styles from './styles.scss'
import manIcon from 'static/man.svg'
// utils
import get from 'lodash/get'
import handleResponseErrors from 'utils/handleResponseErrors'
const cx = classNames.bind(styles)
export default class Deals extends Component {
  static propTypes = {
    showModal: PropTypes.func.isRequired,
    getDeals: PropTypes.func.isRequired,
    startDispute: PropTypes.func.isRequired,
    declineDeal: PropTypes.func.isRequired,
    sendProduct: PropTypes.func.isRequired,
    closeDeal: PropTypes.func.isRequired,
    resources: PropTypes.array.isRequired,
    params: PropTypes.object.isRequired,
    location: PropTypes.object.isRequired,
    total: PropTypes.number.isRequired,
    user: PropTypes.object.isRequired
  }
  constructor (props) {
    super(props)
    this.state = {
      message: ''
    }
  }
  componentDidMount () {
    this.props.getDeals({ deals_type: this.props.location.query.tab })
  }
  componentWillReceiveProps (nextProps) {
    if (nextProps.location.query.tab !== this.props.location.query.tab) {
      this.props.getDeals({ deals_type: nextProps.location.query.tab })
    }
  }
```

```
handleOpenCreateModal = () => {
  if (this.props.user.verified) {
    this.props.showModal(<CreateDealModal />, { title: 'Create deal' })
  } else {
    toastr.info('Info', 'Please verify card in your profile')
  }
}
userCell = (user) => (
  <div className={cx('user-cell')}>
    <img className={cx('user-img')} src={get(user, 'avatar.medium.url') || manIcon}
    /> <span className={cx('user-name')}>{user && user.name}</span>
  </div>
)
handleDeclineDeal = (dealId) => () => {
  this.props.declineDeal(dealId)
    .then(() => {
      toastr.success('Success!', 'Deal has been declined')
    })
    .catch(err => handleResponseErrors(err))
}
handleSendProduct = (dealId) => () => {
  if (this.props.user.verified) {
    this.props.showModal(
      <ChooseDeliveryPeriodModal
        closeModal={this.props.showModal}
        sendProduct={this.props.sendProduct}
        dealId={dealId}
      />,
      { title: 'Select delivery period', size: 'small' }
    )
  } else {
    toastr.info('Info', 'Please verify card in your profile')
  }
}
handleStartDispute = (dealId) => () => {
  const data = { resource: { message: this.state.message }
  } this.props.startDispute(dealId, data)
    .then(() => {
      toastr.success('Success!', 'You started dispute')
    })
    .catch(err => handleResponseErrors(err))
}
handleCloseDeal = (dealId) => () => {
  this.props.closeDeal(dealId)
    .then(() => {
      toastr.success('Success!', 'Deal has been closed!')
    })
    .catch(err => handleResponseErrors(err))
}
handleChangeMessage = ({ target: { value } }) => {
  this.setState({ message: value })
}
```

```
  popoverMessage = (dealId) => (
    <Popover id='popover-message' title='Dispute message'>
      <textarea
        style={{ resize: 'none' }}
        className='form-control'
        name='message'
        rows='5'
        onChange={this.handleChangeMessage}
        value={this.state.message}
      />
      <button style={{ marginTop: 10 }} className='btn btn-success'
onClick={this.handleStartDispute(dealId)}>
        Submit
      </button>
    </Popover>
  )
  renderActions = (deal) => (this.props.user.role ===
    'user' ? (<div>
      {deal.state === 'recipient_selection' && this.props.location.query.tab === 'deals_with_me'
&&
        <button
          className='btn btn-success'
          style={{ marginRight: 10 }}
          onClick={this.handleSendProduct(deal.id)}
        >
          I send product
        </button>}
      {deal.state === 'recipient_selection' &&
        <button className='btn btn-warning'
onClick={this.handleDeclineDeal(deal.id)}>Decline</button>}
      {deal.state === 'pending_of_obtaining' &&
      <OverlayTrigger
        trigger='click'
        rootClose
        placement='bottom'
        overlay={this.popoverMessage(deal.id)}
      >
        <button className='btn btn-
      warning'>Dispute</button> </OverlayTrigger>}
      {deal.state === 'pending_of_obtaining' &&
        <button className='btn btn-success'
onClick={this.handleCloseDeal(deal.id)}>Close</button>}
    </div>)
    : (deal.state === 'disputed'
      && <div>
        <button
          className='btn btn-success' style={{
          marginRight: 10 }}
          onClick={this.handleCloseDeal(deal.id)}
        >
          For Recipient
        </button>
```

```
      <button className='btn btn-primary' onClick={this.handleDeclineDeal(deal.id)}>For Sender</button>
    </div>)
 )
 columns = () => {
  const user = this.props.location.query.tab === 'my_deals'
   ? { label: 'Recipient', key: 'recipient', render: this.userCell }
   : { label: 'Sender', key: 'user', render: this.userCell }
  return ([
   { label: 'Title', key: 'title' },
   user,
   { label: 'Started at', key: 'created_at', date: true },
   { label: 'Updated at', key: 'updated_at', date: true },
   { label: 'Delivery from', key: 'delivery_from', date: true },
   { label: 'Delivery to', key: 'delivery_to', date: true },
   { label: 'Status', key: 'state', humanize: true },
   { label: 'Actions', actions: this.renderActions }
  ])
 }
 handleChangePage = (page) => {
  this.props.getDeals({ deals_type: this.props.location.query.tab, page })
 }
 render () {
  const { resources, params, total, user } = this.props
  return (
   <div>
    {user.role !== 'admin' &&
     <Nav bsStyle='tabs' style={{ marginBottom: 10 }}>
      <LinkContainer to='/dashboard/deals?tab=my_deals'>
       <NavItem>My Deals</NavItem>
      </LinkContainer>
      <LinkContainer to='/dashboard/deals?tab=deals_with_me'>
       <NavItem>Deals with me</NavItem>
      </LinkContainer>
     </Nav>}
    {user.role !== 'admin' && this.props.location.query.tab === 'my_deals'
     && <button
      className='btn btn-success'
      style={{ margin: '10px 0' }}
      onClick={this.handleOpenCreateModal}
     >
      Create new deal
     </button>
    }
    <div className={cx('well', 'tab-content')}>
     <DealTable
      onChangePage={this.handleChangePage}
      columns={this.columns()}
      data={resources}
      total={total}
      params={params}
     />
```

```
      </div>
    </div>
  )
 }
}
```

2) Контеінер DealsContainer

```
import Deals from '../components/Deals'
// redux
import { connect } from 'react-redux'
import { showModal } from 'store/modules/modals'
import { createDeal, getDeals, declineDeal, sendProduct, startDispute, closeDeal } from
'../modules/deals'
const mapActionsCreators = {
 showModal,
 createDeal,
 getDeals,
 declineDeal,
 sendProduct,
 startDispute,
 closeDeal
}

const mapStateToProps = (state) => ({
 resources: state.deals.resources,
 params: state.deals.params,
 total: state.deals.total,
 location: state.router.locationBeforeTransitions,
 user: state.auth.user
})

export default connect(mapStateToProps, mapActionsCreators)(Deals)
```

ДОДАТОК Ц

Код модального віка для створення угоди

1) Компонент CreateDealForm

```
import React, { Component } from 'react'
import PropTypes from 'prop-types'
// components
import { Field } from 'redux-form'
import Input from 'components/FormFields/Input'
export default class CreateDealForm extends Component
 { static propTypes = {
  pristine: PropTypes.bool.isRequired,
  submitting: PropTypes.bool.isRequired,
  handleSubmit: PropTypes.func.isRequired
 }
 render () {
  const { submitting, pristine, handleSubmit } = this.props
  return (
   <form className='form-horizontal left-aligned'
    onSubmit={handleSubmit}> <div className='form-group'>
     <label className='col-sm-12 control-
     label'>Title</label> <div className='col-sm-12'>
      <Field
       className='form-control'
       name='title'
       placeholder='Title'
       component={Input}
       required
      />
     </div>
    </div>
    <div className='form-group'>
     <label className='col-sm-12 control-label'>Email of
     recipient</label> <div className='col-sm-12'>
      <Field
       className='form-control'
       type='email'
       name='email_of_recipient'
       component={Input}
       placeholder='Email of recipient'
       required
      />
     </div>
    </div>
    <div className='form-group'>
     <label className='col-sm-12 control-
     label'>Amount</label> <div className='col-sm-12'>
      <Field
       className='form-control'
```

```
            step='0.5'
            name='amount'
            type='number'
            placeholder='Amount'
            component={Input}
            required
           />
          </div>
        </div>
        <div className='form-group'>
          <label className='col-sm-12 control-
          label'>Description</label> <div className='col-sm-12'>
            <Field
            className='form-control'
            style={{ resize: 'none' }}
            rows={5}
            name='description'
            type='textarea'
            placeholder='Description'
            component={Input}
           />
          </div>
        </div>
        <div className='text-right'>
          <button type='submit' className='btn btn-success' disabled={pristine ||
            submitting}> Save
          </button>
        </div>
      </form>
    )
  }
}
```

2) Контейнер CreateDealForm

```
import CreateDealForm from '../../components/CreateDealForm/index'
import { reduxForm } from 'redux-form'
export default reduxForm({ form: 'create-deal-form' })(CreateDealForm)
```

3) Компонент CreateDeal

```
import React, { Component } from 'react'
import PropTypes from 'prop-types'
// components
import CreateDealForm from './CreateDealFormContainer'
import { toastr } from 'react-redux-toastr'
import { SubmissionError } from 'redux-form'
// utils
import handleResponseErrors from 'utils/handleResponseErrors'
export default class CreateDeal extends Component {
  static propTypes = {
```

```
    createDeal: PropTypes.func.isRequired,
    showModal: PropTypes.func.isRequired
  }
  handleOnSubmit = (data) => {
    return this.props.createDeal({ resource: data })
      .then(() => {
        toastr.success('Success!', 'A Deal has been created!')
        this.props.showModal()
      })
      .catch((err) => {
        handleResponseErrors(err)
        throw new SubmissionError(err)
      })
  }
  render () {
    return (
      <CreateDealForm onSubmit={this.handleOnSubmit} />
    )
  }
}
```

4) Контейнер CreateDealContainer

```
import { connect } from 'react-redux'
import { createDeal } from '../../modules/deals'
import { showModal } from 'store/modules/modals'
import CreateDeal from './CreateDeal'
const mapActionCreators = {
  createDeal,
  showModal
}
export default connect(null, mapActionCreators)(CreateDeal)
```

ДОДАТОК Ш

Редюсер deals.js

```
const SET_DEAL = 'deals/SET_DEAL'
const UPDATE_DEAL = 'deals/UPDATE_DEAL'
const GET_DEALS = 'deals/GET_DEALS' const
LOADING = 'deals/LOADING'
export const createDeal = (data) =>
  (dispatch, getState, client) => {
    dispatch(setLoading(true))
    return client.post('/deals', { data })
      .then((response) => {
        dispatch({ type: SET_DEAL, payload: response.resource })
        dispatch(setLoading(false))
      })
  }
export const getDeals = (params) =>
  (dispatch, getState, client) => {
    dispatch(setLoading(true))
    return client.get('/deals', { params: params })
      .then((response) => {
        dispatch({
          type: GET_DEALS,
          payload: {
            params: { ...defaultParams, params },
            resources: response.resources,
            total: response.meta.total
          }
        })
        dispatch(setLoading(false))
      })
  }
export const declineDeal = (id) =>
  (dispatch, getState, client) => {
    dispatch(setLoading(true))
    return client.put(`/deals/${id}/decline`)
      .then((response) => {
        dispatch({ type: UPDATE_DEAL, payload: response.resource })
        dispatch(setLoading(false))
      })
  }
export const sendProduct = (id, data) =>
  (dispatch, getState, client) => {
    dispatch(setLoading(true))
    return client.put(`/deals/${id}/send_product`, { data })
      .then((response) => {
        dispatch({ type: UPDATE_DEAL, payload: response.resource })
        dispatch(setLoading(false))
      })
  }
```

```
export const startDispute = (id, data) =>
  (dispatch, getState, client) => {
    dispatch(setLoading(true))
    return client.put(`/deals/${id}/start_dispute`, { data })
      .then((response) => {
        dispatch({ type: UPDATE_DEAL, payload: response.resource })
        dispatch(setLoading(false))
      })
  }
export const closeDeal = (id) =>
  (dispatch, getState, client) => {
    dispatch(setLoading(true))
    return client.put(`/deals/${id}/close`)
      .then((response) => {
        dispatch({ type: UPDATE_DEAL, payload: response.resource })
        dispatch(setLoading(false))
      })
  }
const setLoading = (bool) => ({ type: LOADING, payload: bool })
const ACTION_HANDLERS = {
  [SET_DEAL]: (state, action) => ({ ...state, resources: [...state.resources, action.payload] }),
  [UPDATE_DEAL]: (state, { payload }) => ({
    ...state,
    resources: state.resources.map((deal) => deal.id === payload.id ? payload :
  deal) }),
  [GET_DEALS]: (state, action) => ({ ...state, ...action.payload }),
  [LOADING]: (state, action) => ({ ...state, loading: action.payload })
}
const defaultParams = {
  per_page: 15,
  page: 1
}
const initialState = {
  deal: null,
  loading: false,
  resources: [],
  params: defaultParams,
  total: 0
}
export default function counterReducer (state = initialState, action)
  { const handler = ACTION_HANDLERS[action.type]
  return handler ? handler(state, action) : state
}
```