МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ЕЛЕКТРОНІКИ
КАФЕДРА ПРОГРАМУВАННЯ ТА МАТЕМАТИКИ

Пояснювальна записка
до дипломної роботи
бакалавр
(освітньо-кваліфікаційний рівень)

на тему «Розробка серверної частини онлайн платформи для міжнародної торгівлі між комерційними підприємствами»

Виконав: студент 4 курсу, групи ІТ-651
напряму підготовки 6.050103 „Програмна Інженерія ”

_____ Топчій О.Є.
(підпис)

Керівник,
Професор, д.т.н._____ Фесенко Т.М.
(підпис)

Рецензент,
доцент, к.ф.-м.н._____ Ковальов Ю.Г.
(підпис)

СЄВЄРОДОНЕЦЬК
2019 року

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Факультет інформаційних технологій та електроніки
Кафедра програмування та математики
Освітньо-кваліфікаційний рівень бакалавр
Напрям підготовки 6.050103 „Програмна Інженерія ”

ЗАТВЕРДЖУЮ
Завідувач кафедри ПМ,
к.т.н., доцент
_____ Лифар В.О.
«___» _____ 2019 р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ
Топчій Олексію Євгеновичу

1. Тема роботи Розробка серверної частини онлайн платформи для міжнародної торгівлі між комерційними підприємствами.
керівник роботи Фесенко Тетяна Миколаївна
затверджені наказом вищого навчального закладу від “__”__2019 року №___
2. Строк подання студентом роботи 21 травня 2019 р.
3. Вихідні дані до роботи
Об'єктом даної роботи є поняття про електронну торгівлю та побудування веб-додатків на Node.js з використанням Express Framework.
3.1 Літературні джерела:
Ітан Браун – «Веб-розробка з використанням Node та Express»;
Тєдєєв А.А. «Электронна коммерція».
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
4.1 Вступ
4.2 Аналітичний огляд галузі (огляд публічних джерел інформації), з висвітленням наступних питань:
Електронна комерція.
Основні переваги електронної комерції.
Що є торгівельною платформою.
4.3 Основна частина, в якої висвітлити:
Основні вимоги до проектованої платформи.
Побудувати інформаційну модель.
Алгоритми роботи платформи
4.4 Висновки
4.4 Перелік використаних джерел
5. Перелік графічного матеріалу немає

6. Дата видачі завдання 4 лютого 2019 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломної роботи	Строк виконання етапів роботи	**Примітка**
1	Одержання завдання на виконання роботи	01.02.19	
2	Укладання і погодження з керівником плану і етапів виконання роботи	20.02.19	
3	Узагальнення даних літературних джерел, укладання розділу «Аналітичний огляд»	1.03.19	
4	Аналіз шляхів виконання завдання. Вибір і погодження з керівником оптимального шляху	10.03.19	
3	Проектування інфологічної моделі задачі що реалізується.	01.04.19	
5	Укладання та тестування програмного продукту	20.04.19	
6	Укладання, оформлення та погодження пояснювальної записки з керівником	10.05.19	
7	Укладання, оформлення та погодження з консультантом розділу «Охорона праці»	15.05.19	
7	Здача готової пояснювальної записки на кафедру	21.05.19	
8	Укладання доповіді і презентації	01.06.19	

Студент _____

(підпис)

Керівник роботи _____

(підпис)

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ
бакалаврської роботи студента гр. ІТ-651 Топчій О.Є.

Науковий керівник
Професор, д.т.н._____Фесенко Т.М.

Оцінка наукового керівника:_____

Рецензент Ковальов Ю.Г., доцент каф. ПМ СНУ ім. В.Даля
ПІБ, місто роботи, посада

Оцінка рецензента:_____

Кінцева оцінка за результатами захисту:

_____

Голова ЕК,
Зав. Кафедри ПМ
д.т.н, доцент _____Лифар В.О.

РЕФЕРАТ

Текст – 46 с., рис. – 11, табл. – 0, додатків – 1, літературних джерел – 22

У ході виконання даної дипломної роботи було проведено дослідження предметної області, вивчені основні джерела й особливості онлайн платформ електронної комерції.

Під час дослідження було розібрано приклади інших платформ електронної комерції. Розроблено загальну інформаційну схему платформи. Були теоретично виявлені потенційні користувачі платформи та їх проблеми, які зможе вирішити платформа.

Результати роботи дозволять розробити сучасну онлайн платформу, аналогів якої поки що немає у галузі торгівлі сировинними товарами.

Ключові слова: ЕЛЕКТРОННА КОМЕРЦІЯ, ТОРГІВЛЯ, ВЕБ-САЙТ, ПЛАТФОРМА, B2B, КОРИСТУВАЧ, ВЕБ-РОЗРОБКА.

ЗМІСТ

ВСТУП

Актуальність досліджень. Розробка онлайн B2B платформи для міжнародної торгівлі сировинними товарами між комерційними підприємствами має гарну бізнес перспективу, оскільки це надасть користувачам з усього світу можливості для вибору товару та компанії продавцю, купівлі та продажу своїх товарів, створення пропозиції до іншого користувача за своїми умовами та оплати обраного товару. Поява нових компаній-продавців з різних країн приведе до розширення каталогу товарів та загального асортименту сайту. Це розширення, у свою чергу, створить інтерес, який посприяє виникненню попиту на існуючі пропозиції, до платформи в компаніях, які є потенціальними споживачами подібних товарів. Можливість оплати товарів у ETH децентралізованість та прозорість технології Blockchain дозволить зробити процес оплати товару швидким та безпечним.

Таким чином, розробка та запуск цієї платформи приведе до утворення нового децентралізованого світового онлайн ринку сировинних товарів зі своїми партнерами, конкурентами та постійними клієнтами.

Об’єкт досліджень: Створення торгівельної веб-платформи.

Предмет досліджень: Розробка серверної частини веб-сайту на Node.js .

Мета дослідження: Дослідити останні технологічні тренди, аби реалізувати їх у платформі для підвищення якості її роботи.

Завдання дослідження:

а) Теоретично виявити групу потенційних користувачів та їх можливі вимоги до інтерфейсу й функціоналу платформи;

б) Зрозуміти проблеми цільової аудиторії та спланувати способи їхнього вирішення;

в) Розробити легкі й зручні сценарії взаємодії користувачів з

платформою;

г) Реалізувати серверну частину онлайн-платформи, використовуючи власні навички програмування на Node.js .

Методологічна та теоретична основа дослідження: методи веб-розробки на Node.js з використанням Express Framework та MongoDB.

Методи дослідження: пошук та читання наукової літератури й ознайомлення з результатами статистик та наукових досліджень.

Наукова новизна досліджень: це буде єдина сучасна й швидка міжнародна торгівельна платформа, з простим та зручним інтерфейсом, широким спекторм можливостей та використанням передових технологічних трендів.

Практичне значення отриманих результатів. Одержані результати можуть бути використані у процесі створення вдосконаленої торгівельної онлайн-платформи.

РОЗДІЛ 1 АНАЛІТИЧНИЙ ОГЛЯД

Торгівельні онлайн майданчики вже прозвали новим Ельдорадо. Чому? Тому що вони надають можливостей для взаємодії між продавцями та покупцями. Але як розвинути та залучити клієнтів та продавців? У цьому розділі будуть розглянуті основні кроки для розробки не тільки веб-сайту а цілої спільноти, яка буде приносити прибуток.

1.1 Електронна комерція та її переваги

Електронна комерція (від англ. Electronic commerce) — це сфера цифрової економіки, що включає всі фінансові та торгові транзакції, які проводяться за допомогою комп'ютерних мереж, та бізнес-процеси, пов'язані з проведенням цих транзакцій.

У багатьох випадках електронна комерція дозволяє скоротити шлях перепродажу продукту від виробника до споживача. Це можливо завдяки використанню Інтернет-технологій, що надають можливість ефективної прямої взаємодії з кінцевим споживачем, тому компанії можуть виконувати роль, яку традиційно виконували проміжні постачальники. Це також дозволяє накопичувати інформацію про усі продажі та про усіх клієнтів, що у свою чергу дозволяє виконати досконалий бізнес-аналіз та маркетингові дослідження. Це є великою перевагою у конкурентній боротьбі.

Електронне середовище широко використовується для доставки цифрового медіа-контенту (музика, фільми, преса тощо), корисної інформації, освітніх матеріалів, а також компаніями-виробниками програмного забезпечення для його продажу.

Найбільшою перевагою електронної комерції є суттєве зниження витрат

на оформлення угоди та її подальше обслуговування. Тому бізнес-процеси, які можуть бути переведені на електронну основу мають потенціал зниження витрат на них, що у свою чергу призводить до зниження собівартості товару чи послуги. Найвідомішим прикладом здійснення електронної комерції є Інтернет-магазин, який являє собою веб-ресурс з каталогом продукції та можливістю замовлення і оплати товарів, які сподобались покупцю. Все більше компаній у світі впроваджують рішення електронної комерції у своєму бізнесі. Наприклад, всесвітньо відома компанія CISCO не має традиційної мережі дистриб'юторів. Замість того, вона приймає замовлення тільки в електронній формі зокрема зі свого веб-сайту. Інший приклад — це виробники ноутбуків. На своїх веб-сайтах вони розміщують інтерактивні сторінки, де користувач може зконфігурувати собі ноутбук за своїми потребами та оформити замовлення і оплату.

Проект «Торгівельна онлайн платформа для міжнародної торгівлі сировинними товарами» - це веб-сайт для електронної комерції типу B2B.

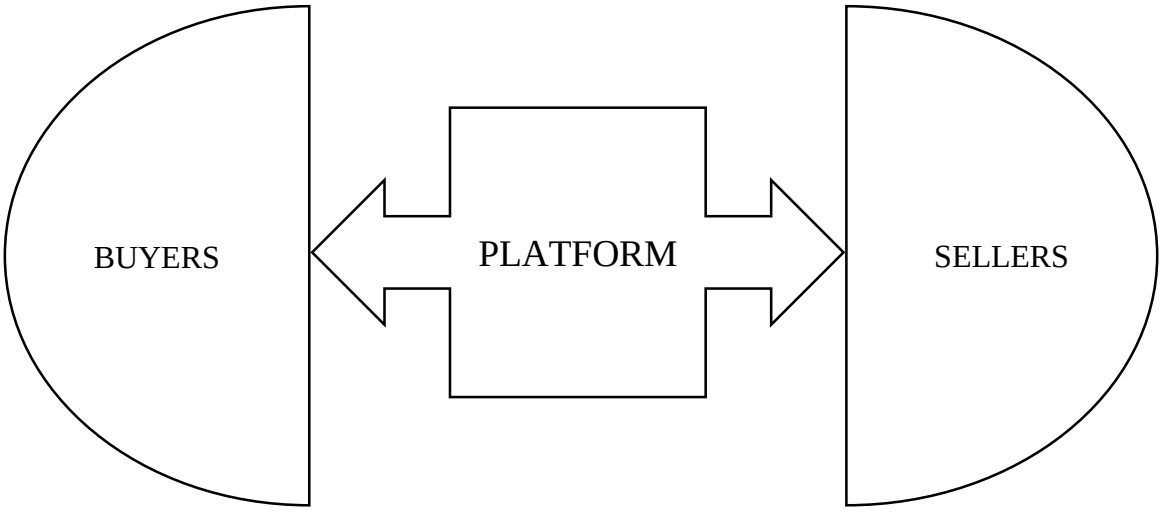
B2B електронна комерція — це електронна комерція між компаніями. Це тип електронної комерції, що має справу з відносинами між видами комерційної діяльності. Приблизно 80% електронної комерції належать до цього типу, і більшість експертів пророкує, що B2B електронна комерція продовжить поширюватися швидше ніж B2C. Загальніші B2B приклади й найкращі практичні моделі: IBM, Hewlett Packard, Cisco та Dell. На сайті моделі B2B можна розміщувати комерційні пропозиції компанії, отримувати відомості від своїх партнерів, постачальників, формувати рахунки для оплати товарів і послуг або навіть укладати контракти. Створення сайтів B2B для компаній дозволяє їм виявляти високу активність на внутрішньому або міжнародному ринку.

1.2 Що є торгівельною платформою

Торгівельна платформа – це веб-сайт на якому покупець та продавець знаходять один одного. Така бізнес модель дуже проста та зручна. Власник онлайн платформи бере процент від продажів та за користування його веб-сайтом.

У такої платформи немає своїх товарів чи послуг. Проте власник торгівельної платформи контролює взаємодію продавців та покупців, виступаючи посередником. З першого погляду, розробка такої платформи – дуже складна задача, але, треба зауважити, що усі найбільші стартапи сьогодні – це подібні платформи.

Одною з найпопулярніших торгівельних платформ сьогодні є Uber. Ідея дуже проста. Ця платформа дозволяє співпрацювати водіям та пасажирам. Компанія була заснована у 2009 році. Зараз це одна з найкрупніших платформ у світі. Що найцікавіше, компанія не має ні одного штатного водія. Цей чудовий приклад того, як платформа може об’єднати у собі клієнтів та продавців. Але без пасажирів не було би й водіїв.



Для створення платформи необхідно 3 компоненти:

- Платформа. Вона об'єднує різні групи користувачів з різними цілями.
- Клієнти. Група користувачів, які хочуть щось купити або скористатися послугами іншої групи користувачів.
- Продавці. Користувачі, які хочуть продати щось або надати послугу покупцям за певну вартість.

Зараз онлайн продажу надають небувалі перспективи для роздрібної торгівлі. Як повідомляє сервіс Statista, в 2015-2016 році кількість людей, які роблять покупки в інтернеті досягло 20 мільйонів.

1.3 Цільова аудиторія та завдання, які буде вирішувати платформа

Вирішувані завдання:

- Комунікація з іншими компаніями або кінцевими споживачами;
- Поширення та надання докладної інформації про товари і послуги;
- Прийом і обробка заявок;
- Спрощення ведення документообігу;
- Зменшення часу на виконання транзакції;
- Створення світового ринку сировинних товарів;

Цільовою аудиторією є менеджери або власники підприємств, які продають або потребують сировинні товари для створення упаковки. Геолокація не має значення, адже платформа спрямована на подолання проблеми бізнесу у замкнутості сфери співпраці адміністративно-територіальними межами. Кожний бажаючий підприємець, або працівник підприємства отримає можливість провести аналіз ринку, продати свій або купити потрібний товар, обравши його з каталогу, або створивши замовлення. Саме це є одною з переваг проекту.

Завдяки цьому кожна компанія-користувач платформи зможе отримати великий асортимент товару, який доповнюється продавцями з усього світу, або величезний ринок збуту власного товару.

1.4 Проблеми аудиторії й шляхи їхнього вирішення

Проблема: Пошук клієнтів – кожна компанія-продавець має свою контактну інформацію, свій веб-сайт, та свій каталог товарів й кожному продавцю потрібна рекламна кампанія, аби поширити цю інформацію.

Вирішення: Аби розповісти клієнтам про себе, треба лише створити профіль своєї компанії, заповнити необхідну інформацію й опублікувати свої товари до каталогу платформи. Тоді кожний зацікавлений покупець побачить інформацію про товар та компанію-продавця. Аби скоріше знайти покупця, кожен продавець отримає можливість переглянути єдиний каталог замовлень від покупців, відсортувати його за категорією, ціною і т.д. й запропонувати свій товар клієнту.

Проблема: Пошук продавців – аби знайти потрібний товар, покупець змушений витратити багато часу на пошук компаній, які займаються продажем товарів цієї категорії.

Вирішення: Створивши профіль своєї компанії, кожен користувач зможе отримати повний каталог товарів та відсортувати його, користуючись системою категорій та фільтрів. Аби не витрачати багато часу на пошук, користувач отримає можливість створити замовлення на товар, яке зможуть переглядати усі продавці й отримати одразу декілька пропозицій від різних продавців, які мають потрібний товар.

Проблема: Територіальне обмеження ринку – асортимент чи кількість можливих клієнтів значно обмежується територіально-адміністративними межами країн, що призводить до замкнутості й ускладнення співпраці та роботи підприємств.

Вирішення: Інтернет торгівля є найкращим вирішенням цієї проблеми, адже користуючись торгівельною платформою покупець й продавець з різних країн світу зможуть знайти один одного й співпрацювати на взаємовигідних умовах. Цей варіант призведе до глобалізації торгівлі сировинними товарами й утворенню нового світового ринку.

Проблема: Недовіра до партнера - купівля та продаж мають певні ризики для обох сторін, саме тому вони повинні довіряти один одному для створення успішної й плідної співпраці.

Вирішення: Користуючись платформою й створюючи нові угоди, компанії зможуть не лише отримати партнерів, а й портфоліо. Кожний новий покупець чи продавець отримає можливість позбавитись недовіри до обраної компанії через перегляд її рейтингу та відгуків про неї, які залишили інші користувачі, які вже уклали угоди й співпрацювали з нею. Платформа також виступає гарантом по проведенню угоди між сторонами й надає можливість до їх спілкування через створення спільного чату.

Проблема: Складність роботи через мережу Інтернет – деякі не зовсім досвідчені користувачі мережі Інтернет можуть мати складності при торгівлі через веб-платформу.

Вирішення: Створення зручного інтерфейсу призначеного для користувачів запобігає виникненню майже усіх можливих складнощів при роботі із платформою. Решту виключить наявність досвідченого онлайн-користувача й режиму навчання користувача.

Проблема: Складність проведення транзакцій – складність міжнародних грошових переказів полягає у тому, що вони витрачають багато часу й тим самим негативно діють на утворення міжнародних комерційних відносин.

Вирішення: Використання технології BC, як засобу для проведення переказу коштів, зменшує час виконання транзакції до декількох хвилин.

1.5 Бізнес-модель

Основною ідеєю платформи для міжнародної торгівлі сировинними товарами є надання продавцям інтернет-платформи для продажу будь-яких товарів. Сама платформа виступає лише в ролі посередника при укладанні договору купівлі-продажу між продавцем і покупцем. Оплата товару і його пересилання відбувається без участі онлайн платформи. Продавці за використання платформи платять внесок, зазвичай складається з збору за виставляння лота і відсотка від ціни продажу. Для покупців використання платформи безкоштовно.

Так як прибуток онлайн платформи безпосередньо залежить від обсягів продажів, здійснених за допомогою цієї платформи, на ній будуть діяти досить ліберальні умови. До продажу дозволені будь-які сировинні товари, що не порушують законодавства тієї країни, в якій зареєстрована компанія-продавець.

Причини ефективності бізнес-моделі:

1. Відсутність географічних бар'єрів - продавці і покупці можуть брати участь в торгах на платформі з будь-якої точки світу, досить мати доступ до Інтернету. Це збільшує число продавців / лотів і кількість покупців / зроблених ставок.

2. Відсутність мовних бар'єрів - участь в аукціонній торгівлі можливо на різних мовах. Багато країн мають власні, локальні філії аукціону,

наприклад Великобританія, Німеччина, Нідерланди, Іспанія, Австралія; цей список може бути продовжений.

3. Відсутність часових рамок - ставки на товари на платформі можна робити 24 години на добу, 7 днів на тиждень. Лоти, в свою чергу, можуть бути виставлені на період до 30 днів - достатній період для пошуку, ознайомлення і покупки.

4. Велика кількість покупців - відвідувачі аукціонів залучені величезним асортиментом різноманітних сировинних товарів, що виставляються за відносно низькими цінами. Також тут можна знайти рідкісні товари, які практично неможливо купити оффлайн.

5. Велика кількість продавців - низькі витрати на розміщення товарів, величезна купівельна аудиторія, простота використання сервісів аукціону платформи також привабливі для продавців. Більш того, продавцем може стати кожен учасник.

6. Мультиплікаційний ефект моделі - зростання кількості покупців призводить до зростання числа продавців, зростання числа продавців стимулює зростання кількості покупців. Що і гарантує розвиток платформи в усьому світі.

РОЗДІЛ 2 ІНФОРМАЦІЙНА МОДЕЛЬ ПРЕДМЕТНОЇ ГАЛУЗІ

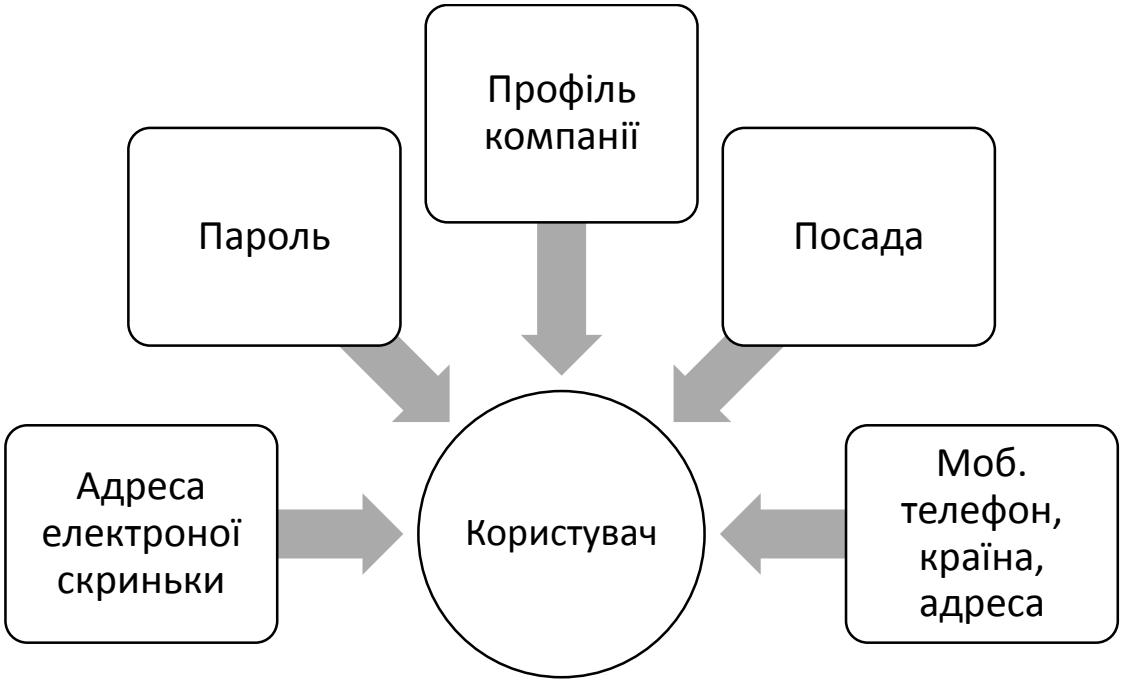
Потрібно розробити швидкий, зручний та надійний засіб для публікації товарів, їхньої покупки, укладання угод між сторонами, будування та розвитку Інтернет бізнесу, створенню онлайн ринку та його адміністрування.

У цьому розділі будуть розглянуті основні сутності даних та інформаційні моделі, які будуть використовуватися для роботи платформи.

2.1 Користувач

Користувач – це людина, яка пройшла реєстрацію на платформі, аби скористатися можливостями, які вона надає для розвитку бізнесу й отримання прибутку. Користувач є представником свого підприємства й взаємодіє з іншими користувачами через профіль своєї компанії.

Для платформи, аби ідентифікувати користувача й отримати канал для зв’язку тільки з цим користувачем, він повинен залишити свій адрес електронної скриньки, якій подальше буде також використовуватись для нотифікації, й вказати пароль до аккаунту. Адже користувач буде взаємодіяти з іншими користувачами через профіль своєї компанії, модель користувача повинна також містити ідентифікатор його корпоративного профілю та його посаду у цій компанії. Для випадків, коли потрібна ідентифікація користувача, який використовує корпоративний профіль, потрібна наявність фотографії користувача й деяких, також необов’язкових даних, таких як: номер мобільного телефону, країна й адреса.



Зображення інформаційної моделі користувача

2.2 Профіль компанії

Вид електронної комерції B2B має на увазі відносини на рівні підприємство до підприємства, саме тому профіль компанії – це «обличчя» або «персона» користувача, від імені якої відбуваються усі події та взаємодії на платформі пов’язані з ним.

Профіль компанії – це комплексна сутність, побудована з трьох частин:

- Основний профіль
- Фінансова інформація
- Інформація про користувачів
- Гаманець

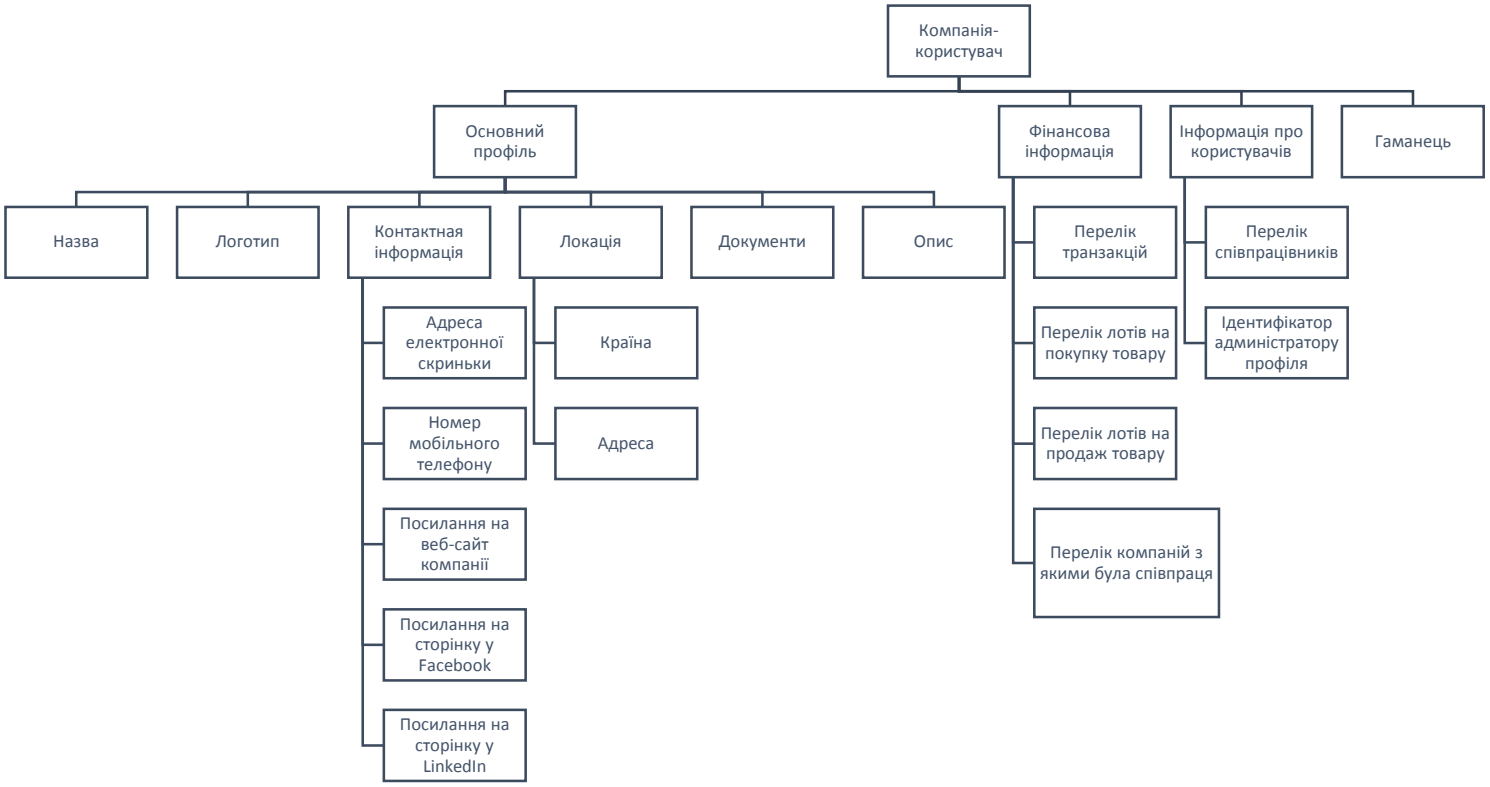
Основний профіль – це публічно доступний розділ моделі компанії, який містить загальну інформацію про компанію та її діяльність. Цей розділ повинен містити у собі такі необхідні дані, як назва компанії та її логотип, які

необхідні для ідентифікації компанії. Опис компанії потрібен для усвідомлення галузі роботи компанії й орієнтації користувача під час вибору торгівельних партнерів. Необхідно, щоб основний профіль компанії також містив контактну інформацію компанії на випадок складної ситуації, дезинформації чи відсутності зв'язку з компанією через мережу Інтернет. Контактна інформація компанії повинна містити у собі наступні дані: номер мобільного телефону, обов'язково; посилання на веб-сайт, необов'язково; адрес корпоративної електронної скриньки, обов'язково; посилання на сторінки у соціальних мережах Facebook та LinkedIn, необов'язково. Для усвідомлення про географічне положення та розрахунків логістичних процесів під час вибору торгівельного партнеру та при плануванні угоди, потрібна інформація про географічне положення компанії або її головного офісу, яке буде складатися з даних про країну знаходження й точну адресу. Для підтвердження легальності комерційної діяльності компанії, користувач, який створює профіль, повинен надати декілька документів у свій профіль компанії, які будуть свідчити про це, й будуть доступні іншим компаніям-користувачам платформи.

Фінансова інформація – розділ профілю компанії, який є приватним й доступним тільки для власника профілю й адміністратора платформи. Він містить у собі інформацію про транзакції у вигляді переліку інформації про перекази коштів у фіатній та крипто валюті. Також цей розділ має перелік лотів на покупку або продаж товарів на платформі й список компаній-користувачів, з якими було укладено угоду на покупку або продаж товару.

Інформація про користувачів – приватний розділ профілю компанії, який містить у собі інформацію про користувачів, які працюють у цій компанії. Він має у собі перелік співробітників, які також пройшли реєстрацію на платформі для виконання своїх обов'язків на платформі й ідентифікатор адміністратора профілю компанії, який безпосередньо має над ним контроль.

Гаманець – публічний внутрішній ідентифікатор гаманця компанії у ВС мережі Ethereum для зберігання коштів й проведення відправлення або отримання оплати у разі укладення угод з іншими компаніями-користувачами.



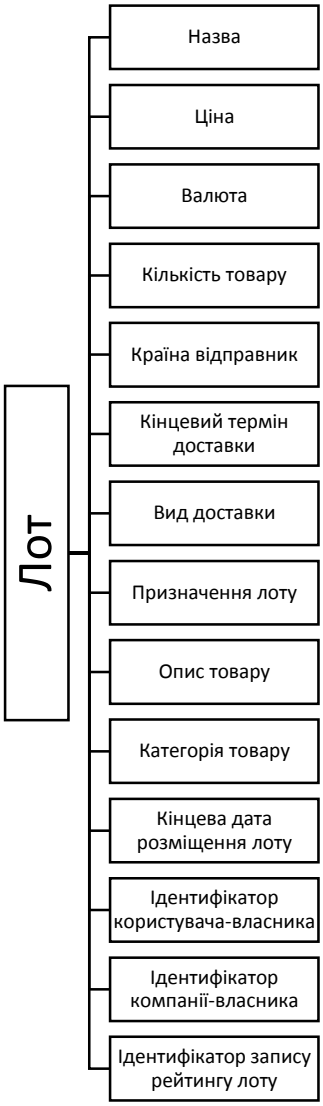
Зображення інформаційної моделі профілю компанії

2.3 Лот

Лот – це пропозиція, яка є складовою переліку пропозицій на продаж або купівлю певних товарів, які було опубліковано користувачами платформи за певними категоріями.

Перш за все лот повинен мати свою назву й інформацію про власника. Інформація про власника – це ідентифікатор каристувача, який виконва публікацію лоту, й ідентифікатор компанії співпрацівником якої він є. Для того щоб відрізняти лот для продажу від лоту для купівлі потрібно вказати його призначення: продаж або купівля. Аби надати ясності до того, що

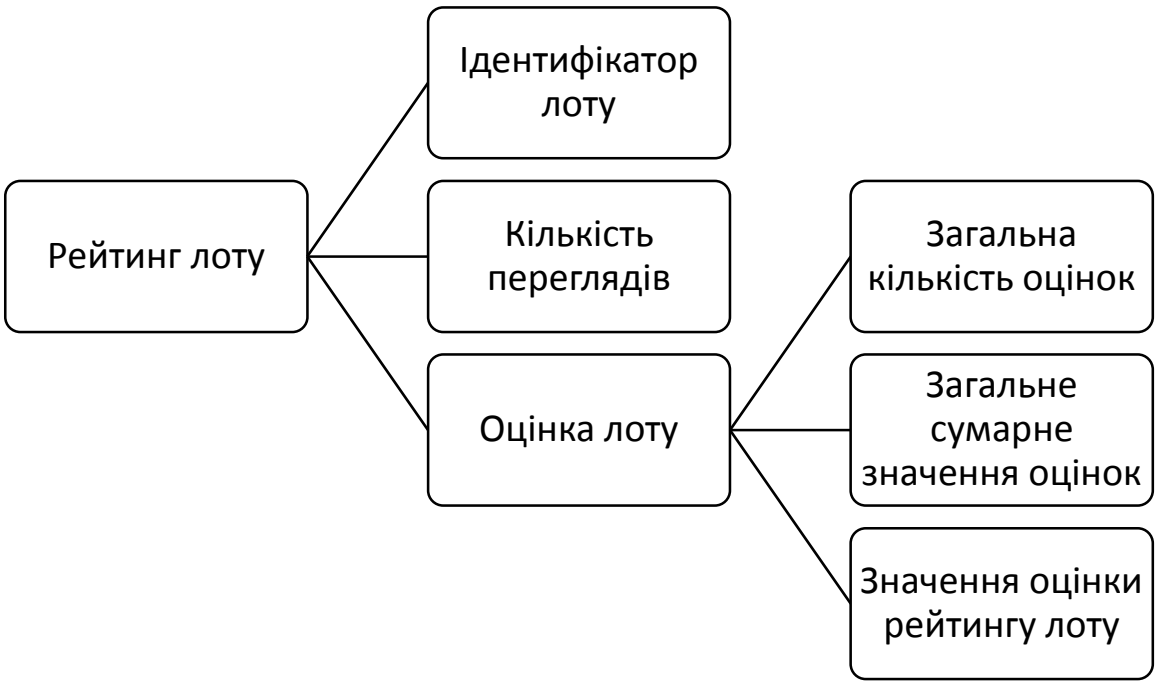
продається або купляється, користувач, який виконує публікацію, також повинен додати опис товару у вигляді розгорнутого тексту, й ідентифікатору категорії для того, щоб платформа змогла віднести лот до певної групи товарів й відображати його у відповідному розділі. Так як мається на увазі саме міжнародна торгівля між підприємствами, обов’язково потрібно, щоб користувач вказав країну відправлення, останній термін доставки й один з видів міжнародної доставки, вказавши один з доступних для вибору DT. Для зручності укладення угод між підприємствами з різних країн, грошові перекази повинні бути можливі для виконання у декількох валютах, тому користувач повинен вказати суму до сплати, валюту, у якій будуть сплачуватися кошти у разі укладання угоди й кількість товару за вказану ціну. Для того, щоб візуалізувати товар то якого був створений лот, потрібно щоб лот містив у собі декілька зображень товару. Для збільшення довіри до компанії, яка виставляє лот, потрібно докласти до лоту декілька підтверджуючих документів. Також, за своїм бажанням, користувач може вказати кінцеву дату розміщення лоту.



Зображення інформаційної моделі лоту

2.4 Рейтинг лоту

Кожен лот має рейтинг у вигляді окремого комплексного запису. Цей запис має ідентифікатор лоту, до якого він належить, аби віднести цей запис до потрібного лоту. Для визначення потрібності лоту, кожен запис про рейтинг потрібен мати у собі кількість переглядів від користувачів, які не є власниками лоту або співпрацівниками компанії власника лоту. Сам рейтинг лоту, від 1 до 5 зірок, який виставляється користувачами платформи, які також не є власниками або співпрацівниками компанії власника, складається з кількості оцінок, їхнього сумарного значення та середньо-арифметичного значення оцінки цього лота.



Зображення інформаційної моделі рейтингового запису лоту

2.5 Транзакція

Транзакція – операція списання коштів з електронного рахунку користувача, запис про яку повинен зберігатися у базі даних платформи для надання користувачу можливості дослідження історії його витрат та доходів.

Транзакція повинна мати, у першу чергу запис про користувача, з балансу якого були списані кошти, який складається з ідентифікатору аккаунту користувача й ідентифікатора профілю його компанії, від імені якого був здійснений переказ. Також є необхідним запис про користувача, на рахунок якого надійшли ці кошти, який складається з ідентифікатору його аккаунту та ідентифікатору профіля його компанії, від імені якого були отримані на рахунок ці кошти. Про кількість коштів, які здійснили участь у переказі повинні відповідати 3 записи:

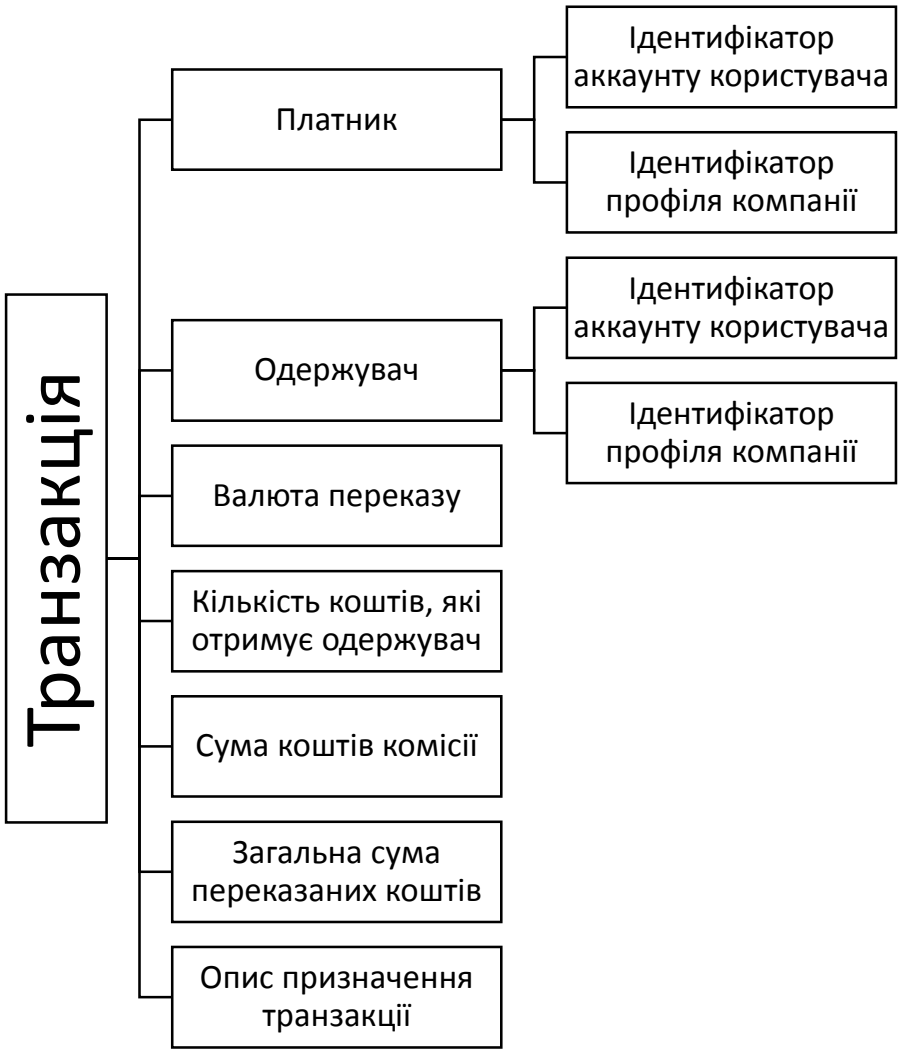
- Кількість – кількість коштів, яка є еквівалентною до ціни, яка була

сформована при укладанні угоди;

- Сума комісії – комісія, значення якої подано у кількості коштів, яку платформа збирає з поданої транзакції;

- Загальна кількість – сума ціни угоди й суми комісії разом для надання інформації про загальну кількість коштів, яка була списана з електронного рахунку користувача, який виконує роль платника.

Опис транзакції - інформація про мету з якої були списані кошти, він є інформацією про угоду, яка була оплачена, згідно з умовами, укладеними обома сторонами. Кожний запис про транзакцію повинен мати у собі також відмітку про валюту, у якій був здійснений переказ коштів.

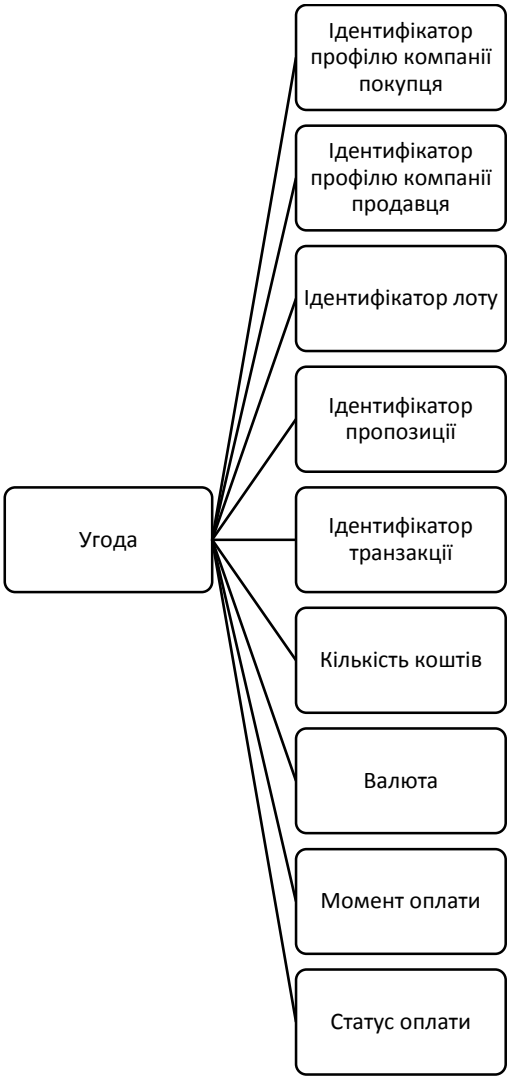


2.6 Угода

Угода – дія між двома компаніями-користувачами направлена на укладання договору про продаж деякого товару одною зі сторін іншій. Інформація про укладені угоди повинна зберігатися у базі даних платформи для надання користувачу можливості для переглядання історії про його взаємодію з іншими компаніями та її подальшого дослідження й аналізу.

Запис про угоду між двома компаніями-користувачами обов’язково повинен містити у собі інформацію про компанію-покупця товару, вона ж є платником, й про компанію-продавця, вона ж є отримувачем. На платформі угода між компаніями не може бути укладена без участі лоту, адже він є зв’язуючою ланкою між двома сторонами. Якщо угода була укладена на основі спеціальної пропозиції від одної компанії до іншої на основі лоту, то запис про угоду повинен містити у собі ідентифікатор спеціальної пропозиції. Для встановлення зв’язку між переказом коштів та угоди, для якої вони були призначені, у записі про угоди буде зберігатися ідентифікатор транзакції. Інформація про кількість переказаних коштів та валюту, у якій був виконаний переказ, у записі про угоду потрібна для її зручного видання користувачеві при запиті історії угод з іншими компаніями.

При укладанні угоди, платник – компанія-покупець – отримає можливість обрати момент оплати: передоплата, або оплата при отриманні товару. Саме тому запис про угоду між сторонами повинен містити у собі інформацію про обраний момент оплати та статус оплати, який може бути тільки двох станів: «оплачено» та «неоплачено».



Зображення інформаційної моделі угоди

РОЗДІЛ 3 ВИБІР МЕТОДІВ ТА РЕАЛІЗАЦІЯ ПОСТАВЛЕНИХ ЗАДАЧ

Коли описані основні задачі, які призвана вирішити онлайн платформа для міжнародної торгівлі сировинними товарами між комерційними підприємствами, та побудована інформаційна модель основних елементів веб-сайту, процес розробки переходить до вибору методів реалізації та їх використання.

У цьому розділі будуть розглянутий процес реалізації методів платформи, основні чинники й вимоги до її функціоналу. Програмна частина серверу буде розроблена на Node.js з використанням Express Framework та MongoDB.

3.1 Планування архітектури платформи

Одною з основних вимог до платформи є зручність користування й легкість масштабування. Легкість використання полягає у можливості користування платформою з будь-якого девайсу, будь-яким чином. Різні користувачі користуються мережею Інтернет по-різному. Наприклад, користувач А віддає перевагу роботі через браузер на своєму персональному комп’ютері, а користувач В бажає мати можливість користуватися платформою через мобільний додаток на смартфоні або планшетному комп’ютері. Розробка двох паралельних систем для вирішування однієї й тієї ж задачі й розроблення єдиної екосистеми для обох видів користувачів є поганим рішенням, тому що це може призвести до ряду проблем при їх синхронізації й подовженню загального часу, який потрібен на реалізацію. До того ж може з’явитися користувач С, якому зручніше побудувати свій додаток для роботи з

платформою. Тоді треба розробити єдину систему, яка буд легкою для перебудовування деяких її областей під час адаптації під користувачів й надасть можливість користувачам усіх категорій використовувати платформу таким способом, який є для них найбільш зручнішим. Для цього було вирішено розробити додаток, користуючись архітектурним стилем «REST».

«REST» — це архітектурний стиль для розподілених гіпертекстових систем. Дані повинні передаватися у вигляді невеликої кількості стандартних форматів (наприклад, HTML, XML, JSON). Будь-який REST протокол (HTTP в тому числі) повинен підтримувати кешування, не повинен залежати від мережевого прошарку, не повинен зберігати інформації про стан між парами «запит-відповідь». Стверджується, що такий підхід забезпечує масштабовність системи і дозволяє їй еволюціонувати з новими вимогами.

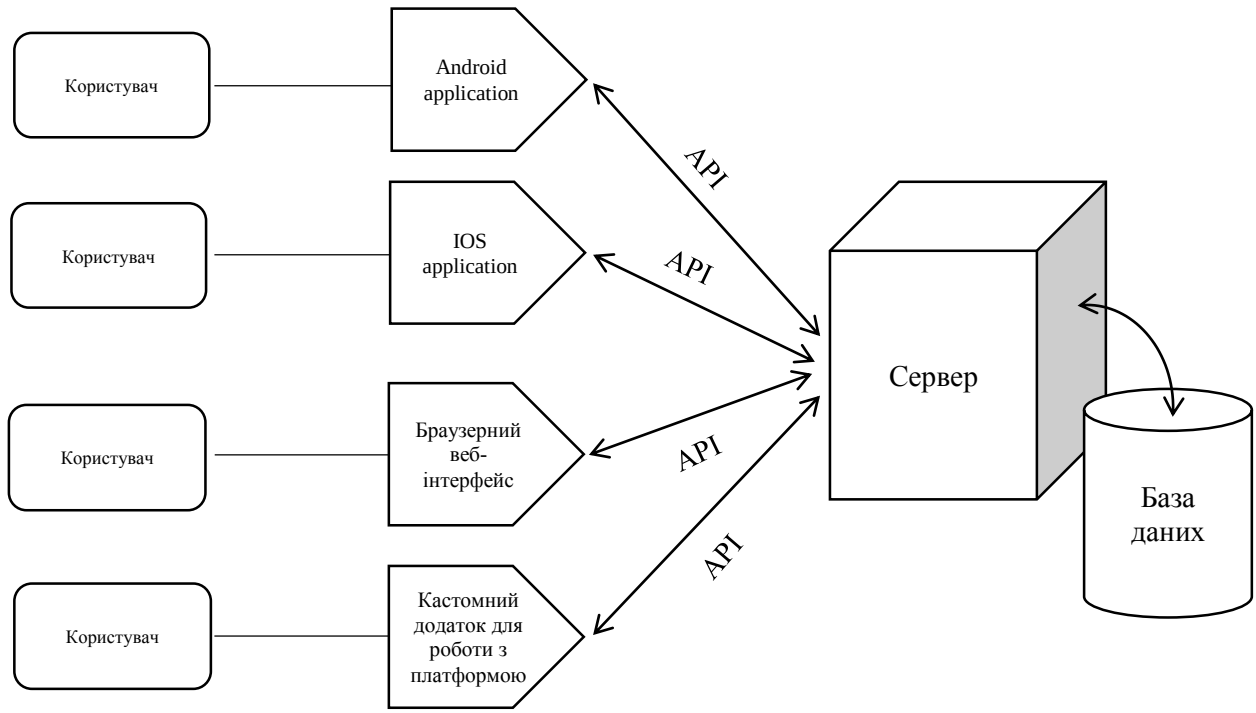
Для створення системи, яка буде невибагливою до методу використовування, було вирішено використовувати схему роботи «клієнт-сервер» й розробити для цього єдиний спеціальний прикладний програмний інтерфейс (далі API).

Прикладни́й програ́мний інтерфе́йс (англ. Application Programming Interface, API) — набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення. Спрощено - це набір чітко визначених методів для взаємодії різних компонентів. API надає розробнику засоби для швидкої розробки програмного забезпечення. API може бути для веб-базованих систем, операційних систем, баз даних, апаратного забезпечення, програмних бібліотек.

«Клієнт» - програмний додаток, який, використовуючи протокол HTTP й розроблений API, буде встановлювати зв’язок з сервером, робити запити та

отримувати та обробляти відповіді для синхронізації даних й виконання дій від імені користувача.

«Сервер» - HTTP сервер, програмний додаток, який буде обробляти вхідні запити, обробляти їх, контролювати доступ до виконання певних дій й отримання інформації, виконувати певні дії за запитом та працювати з базою даних платформи.



Зображення схеми роботи «клієнт-сервер»

Такий підхід об’єднує запити відправлені з різних клієнтів й дозволяє серверу обробляти їх, незалежно від типу платформи з якої було виконано запит. Створювання єдиного способу роботи з сервером для усіх платформ надасть зручності у використанні для користувачів, легкості у розробці й масштабуванні.

3.2 Реєстрація, авторизація та відновлення паролю облікового запису

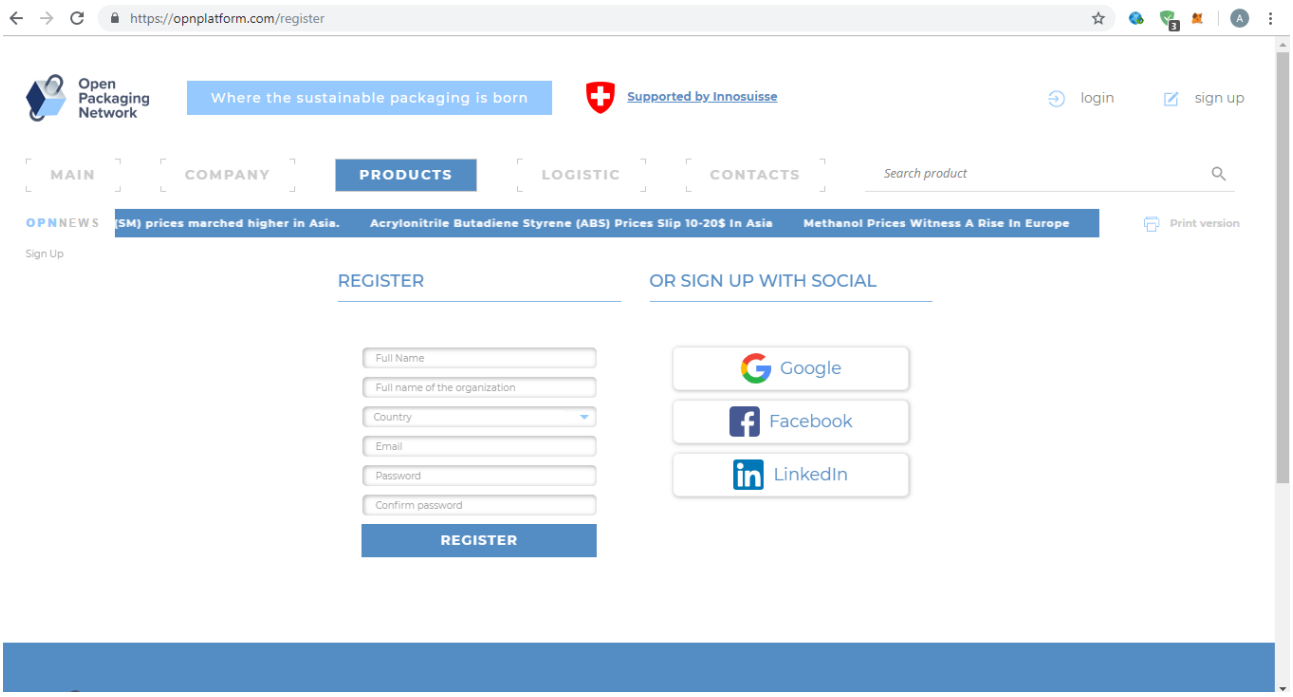
3.2.1 Реєстрація

Реєстрація є необхідною для користування повними можливостями онлайн платформи для міжнародної торгівлі сировинними товарами. Реєстрація – це також перша процедура, яку виконає користувач, тому вона повинна бути максимально простою, зручною та легкою. Онлайн-платформа представляє два способи реєстрації, які будуть описані у наступних підрозділах.

3.2.1.1 Стандартна реєстрація через електронну скриньку

При такому способі реєстрації, користувач повинен перейти на сторінку реєстрації й вказати у відповідних полях наступну інформацію:

1. Адресу електронної скриньки;
2. Своє ім’я та прізвище;
3. Назву своєї компанії;
4. Країну де він знаходиться;
5. Пароль для свого аккаунту на платформі та підтвердження паролю;



Вигляд сторінки реєстрації користувачів онлайн-платформи

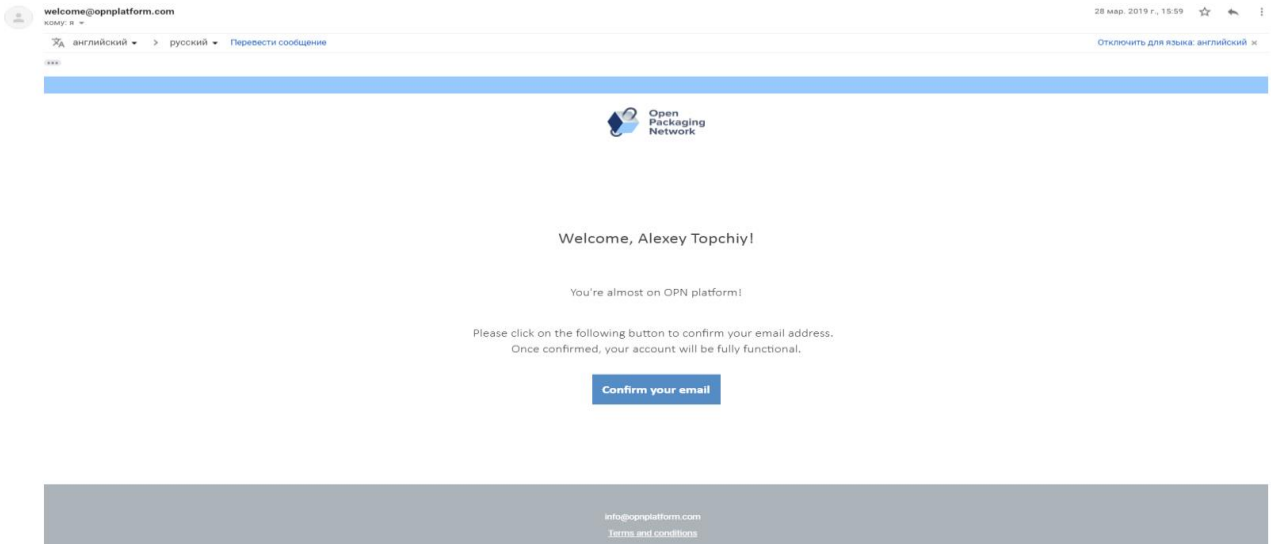
Після того, як користувач вказав необхідні дані, треба натиснути кнопку «Register» для того, щоб відправити дані до сервера.

Наступна послідовність дій серверу:

1. Верифікація отриманих даних;
2. Створення та збереження облікового запису у базі даних;
3. Створення та збереження об’єкту чат-боксу;
4. Створення запису про користувача у CRM Hubspot;
5. Перевірка адреси електронної пошти та створення запису про компанію у CRM Hubspot, якщо її домен є корпоративним;
6. Створення порожнього профілю компанії;
7. Відправлення листа з посиланням на підтвердження адреси електронної пошти, яке є необхідним для завершення процедури реєстрації.

CRM Hubspot буде використовуватися для полегшення задач, які вирішує відділ маркетингу. Однією з гарних функцій цієї системи контролю є можливість отримання публічних даних про компанію, якими вона володіє. Це буде

використано у розробці подальшого функціоналу й розглянуто а наступних розділах. Створення запису про користувача необхідно для занесення даних про його активність на платформі й подальшої синхронізації та аналітики.



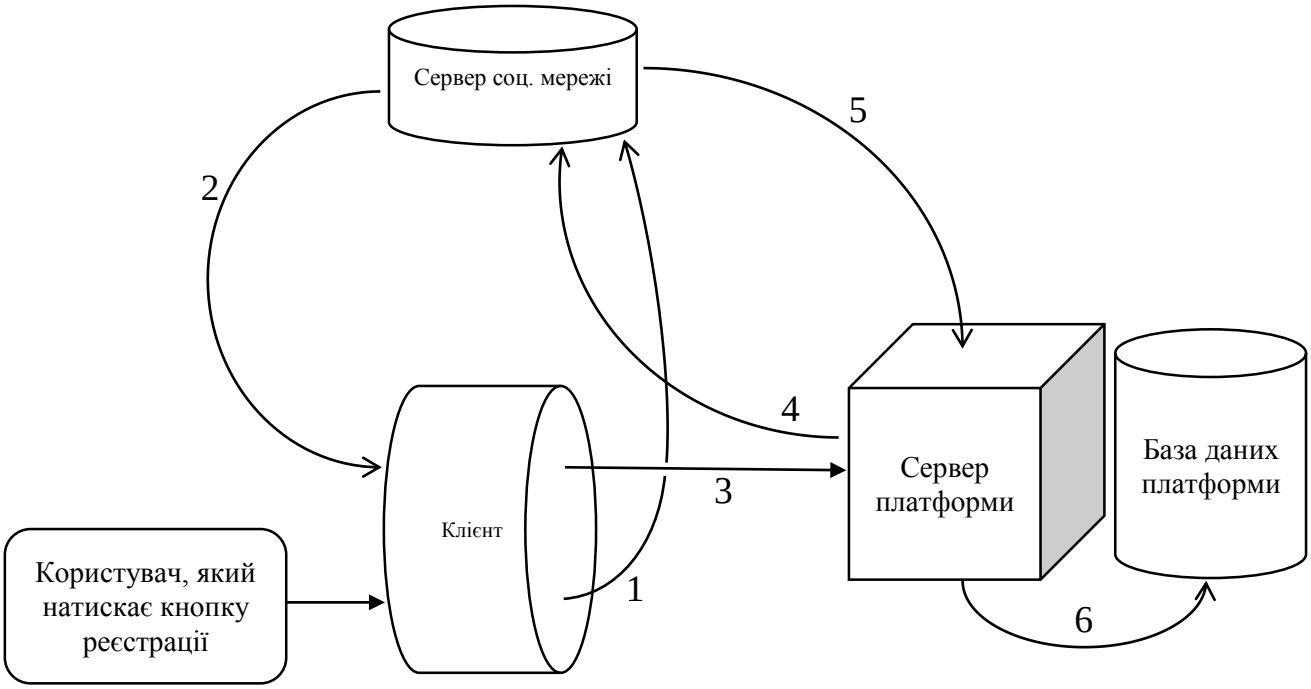
Електронний лист з посиланням для підтвердження адреси електронної пошти, який отримає користувач

Після підтвердження адреси, користувач буде переведений на сторінку авторизації для продовження роботи з платформою.

3.2.1.2 Реєстрація через соціальні мережі

Для спрощення процесу реєстрації, користувач отримає можливість завести обліковий запис, скористувавшись своїм обліковим записом у соціальних мережах Facebook, LinkedIn або використовуючи реєстрацію через Google пошту. Це найлегший спосіб реєстрації, тому що користувач повинен лише натиснути кнопку для реєстрації, й дозволити доступ до своїх даних платформі у спливаючому вікні. Після цього, використовуючи спеціальний API метод, клієнт відправить на сервер код авторизації, який сервер використає для отримання даних профілю у відповідній соціальній мережі.

Таким чином, цей спосіб реєстрації можна описати наступною схемою:



Реєстрація через соціальні мережі покроково:

1) *Користувач переходить до сторінки з авторизацією у соціальній мережі, проходить її й дозволяє доступ до аккаунту;*
2) *Сервер соціальної мережі відповідає з кодом авторизації;*
3) *Клієнт посилає отриманий код авторизації на сервер платформи використовуючи спеціальний API метод;*
4) *Сервер платформи посилає запит з отриманим кодом авторизації на сервер соціальної мережі, щоб отримати інформацію з профіля користувача;*
5) *Сервер соціальної мережі відповідає на запит з інформацією про профіль користувача;*
6) *Сервер платформи оброблює отриману інформацію про профіль користувача, виконує процедуру як при стандартній реєстрації через адресу електронної пошти, тільки без процедури підтвердження адреси й зберігає усе у базі даних.*

Завдяки такій схемі роботи участь користувача у процесі реєстрації зводиться

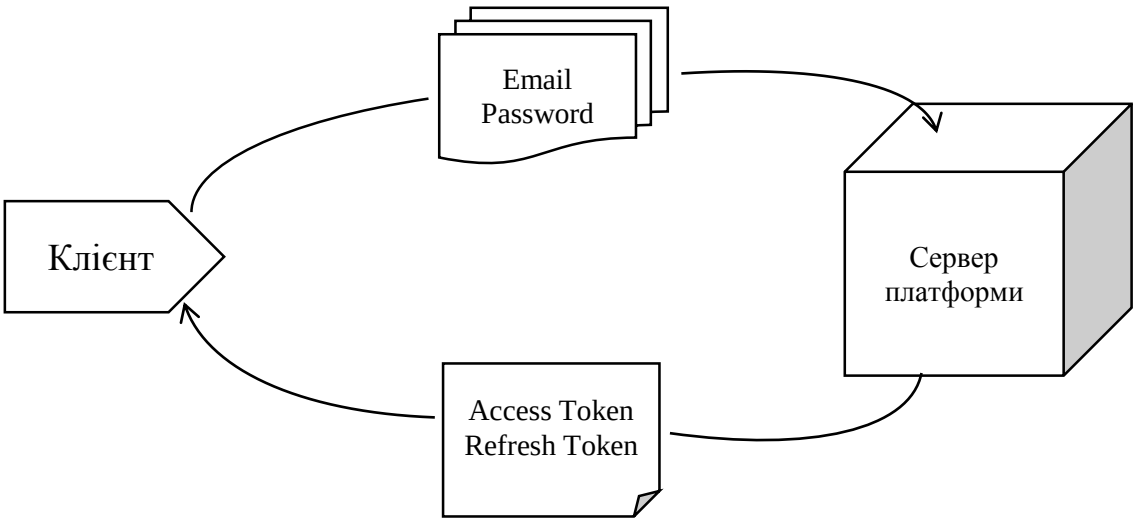
практично до двох або трьох простих дій, що сприяє задоволенню бажань користувача й зменшенню загального часу, витраченого на виконання процедури.

3.2.2 Аутентифікація

Архітектурний стиль «REST» не передбачає зберігання поточного стану використання додатку користувачем. Це означає, що додаток, який розроблюється за цим архітектурним стилем, не може використовувати сесії або кукі-файли. Кожен новий запит повинен мати ідентифікатор на сервері для проходження контролю доступу та привілеїв. Тобто кожен новий запит від клієнта повинен бути заново аутентифікований, а вже потім, у разі успішної аутентифікації, його буде оброблено та дія, яку запросив користувач через клієнт, буде виконано. Для побудування такої схеми контролю доступу та привілеїв було вирішено скористатися технологією авторизації з використанням API ключа.

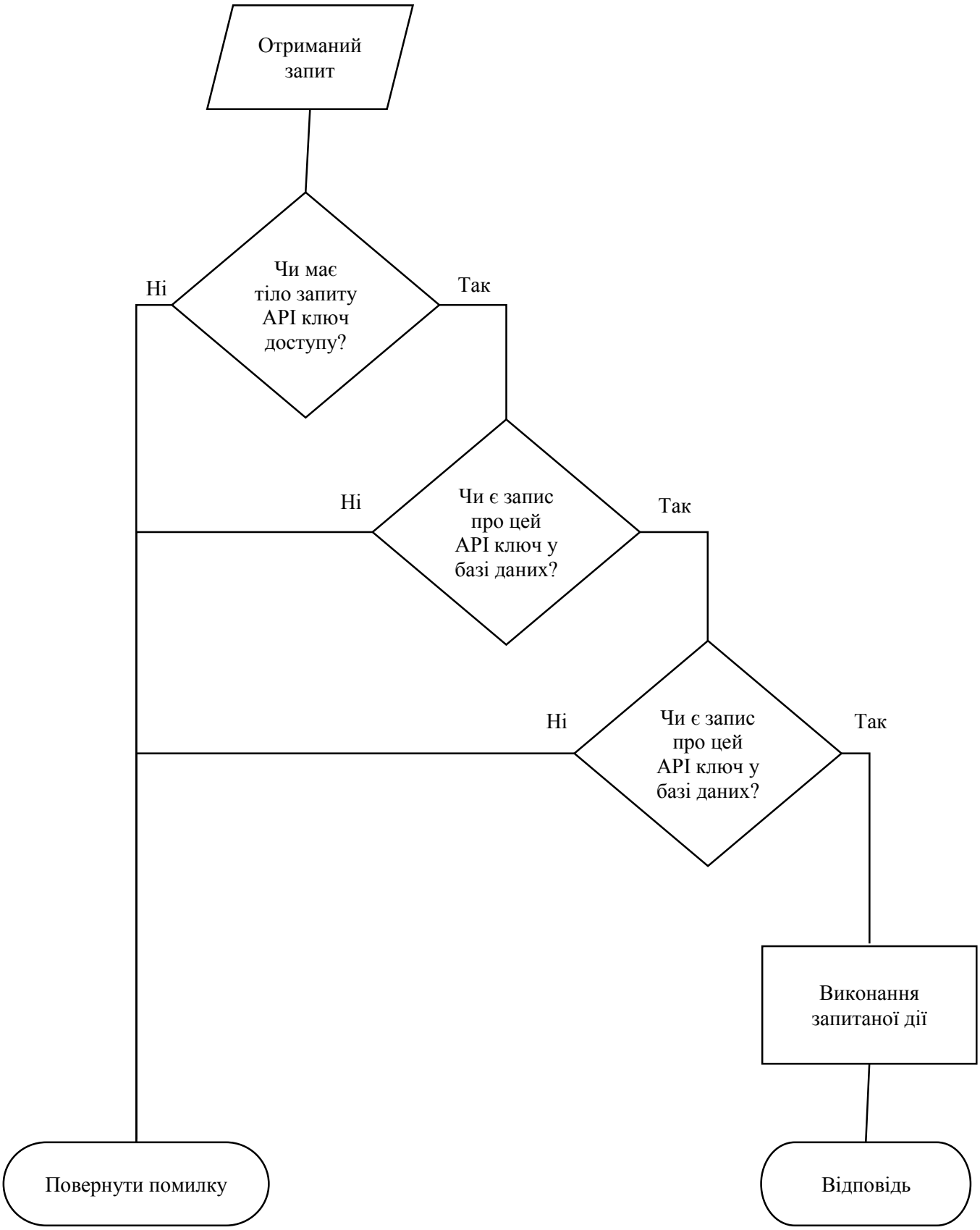
Ключ інтерфейсу прикладного програмування (ключ API) - це код, який передається комп'ютерними програмами, що викликають інтерфейс прикладного програмування (API), для аутентифікації викликаючої програми, її розробника або користувача на веб-сайті. Ключі API використовуються для відстеження та контролю використання API, наприклад, для запобігання не коректного використання або зловживання API. За цією схемою роботи, якщо користувач був зареєстрований через адресу своєї електронної пошти, то він повинен надіслати через клієнт адресу електронної пошти й пароль облікового запису, тоді, якщо було надані правильні дані, сервер надішле у відповідь пару API ключів –доступу й відновлення доступу. Ключ доступу потрібен для аутентифікації кожного наступного запиту до сервера, виконання якого потребує від користувача бути авторизованим у системі. На випадок, якщо

ключ був перехоплений зловмисником, він має строк придатності, який дорівнює одній добі. Для того щоб не потрібно було повторювати авторизацію кожної доби, клієнт використовує API для відновлення ключа доступу, для якого потрібен ключ відновлення доступу. Ключ відновлення доступу має строк придатності рівний одному тижню. Після того, як строк придатності ключа відновлення доступу буде закінчений, користувач потрібен відновити пару API ключів, виконавши процедуру авторизації.



Зображення процесу авторизації, використовуючи адресу пошти та пароль

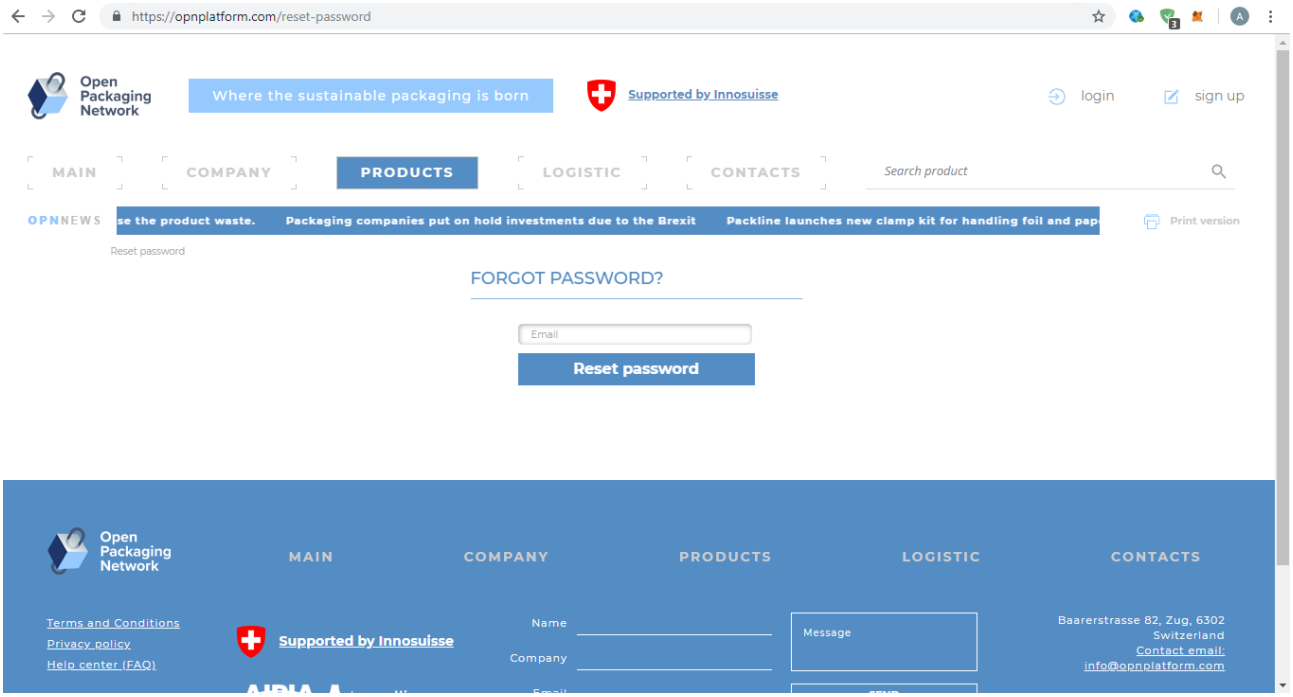
Після проходження процедури авторизації та отримання пари API ключів, кожен наступний запит, для виконання якого сервер вимагає проходження аутентифікації, повинен мати у своєму тілі ключ доступу. Таким чином, процес опрацьовування кожного такого запиту сервером можна зобразити такою схемою:


Отриманий запит
Чи має тіло запиту API ключ доступу?
Ні
Так
Чи є запис про цей API ключ у базі даних?
Ні
Так
Чи є запис про цей API ключ у базі даних?
Ні
Так
Виконання запитаної дії
Повернути помилку
Відповідь

3.2.3 Відновлення паролю до облікового запису

Інколи трапляється ситуація, коли користувач забув пароль від свого облікового запису, або пароль було скомпрометовано. Тоді для забезпечення безпеки та конфіденційності даних облікового запису користувача, потрібно змінити пароль.

Для того, щоб виконати скидання та відновлення паролю, користувач повинен перейти на відповідну сторінку й надати адрес своєї електронної скриньки й натиснути клавішу.



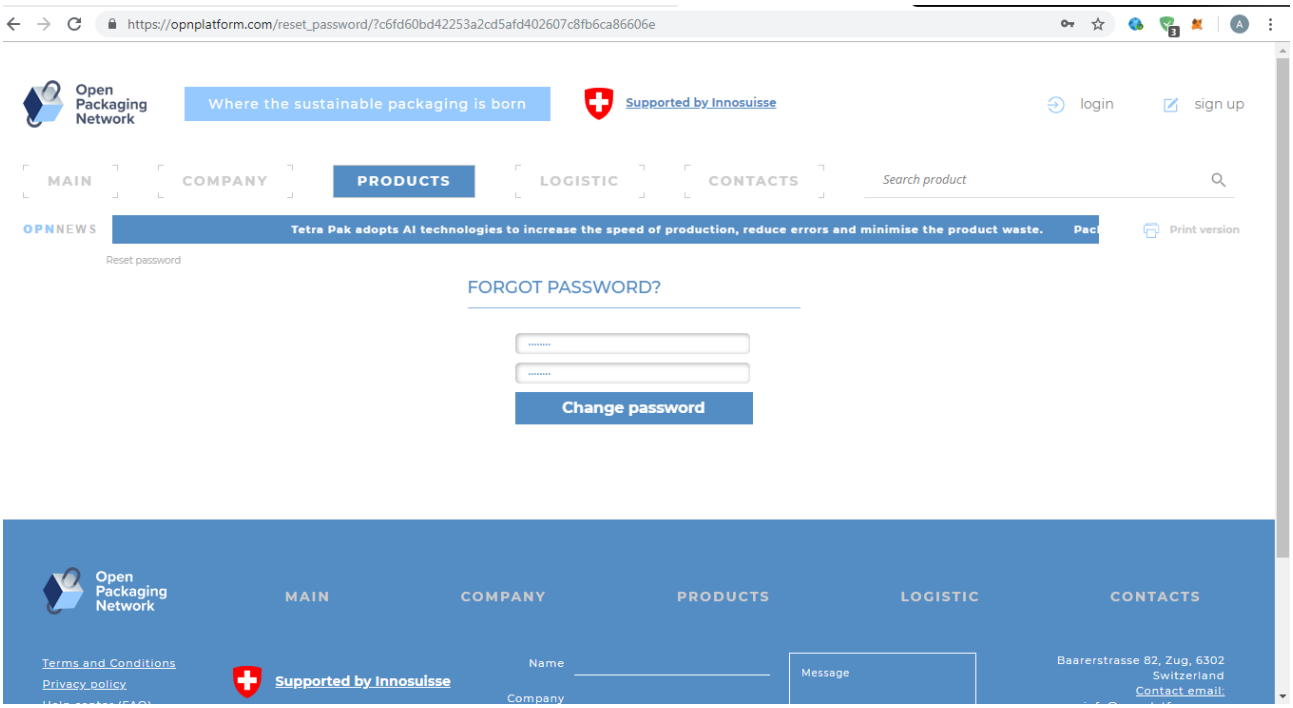
Зображення сторінки для виконування скидання паролю

Після відправлення адреси, сервер буде обробляти запиту наступній послідовності:

1. Перевірка наявності облікового запису з наданою адресою електронної пошти;
2. Якщо таки запис існує, то генерує посилання на сторінку для відновлення паролю, інакше повертає помилку;
3. Посилає користувачу персонального листа зі згенерованим посиланням.

Аби відновити свій пароль, користувач повинен отримати й прочитати листа від онлайн платформи. Потім перейти за посиланням з листа й ввести новий пароль та підтвердження нового паролю до відповідних полів. Після натиснення клавіші, введенні дані будуть відправлені до сервера для подальшої обробки. Якщо дані були валідні, то сервер створить новий хеш паролю й збереже його до бази даних замість старого паролю.

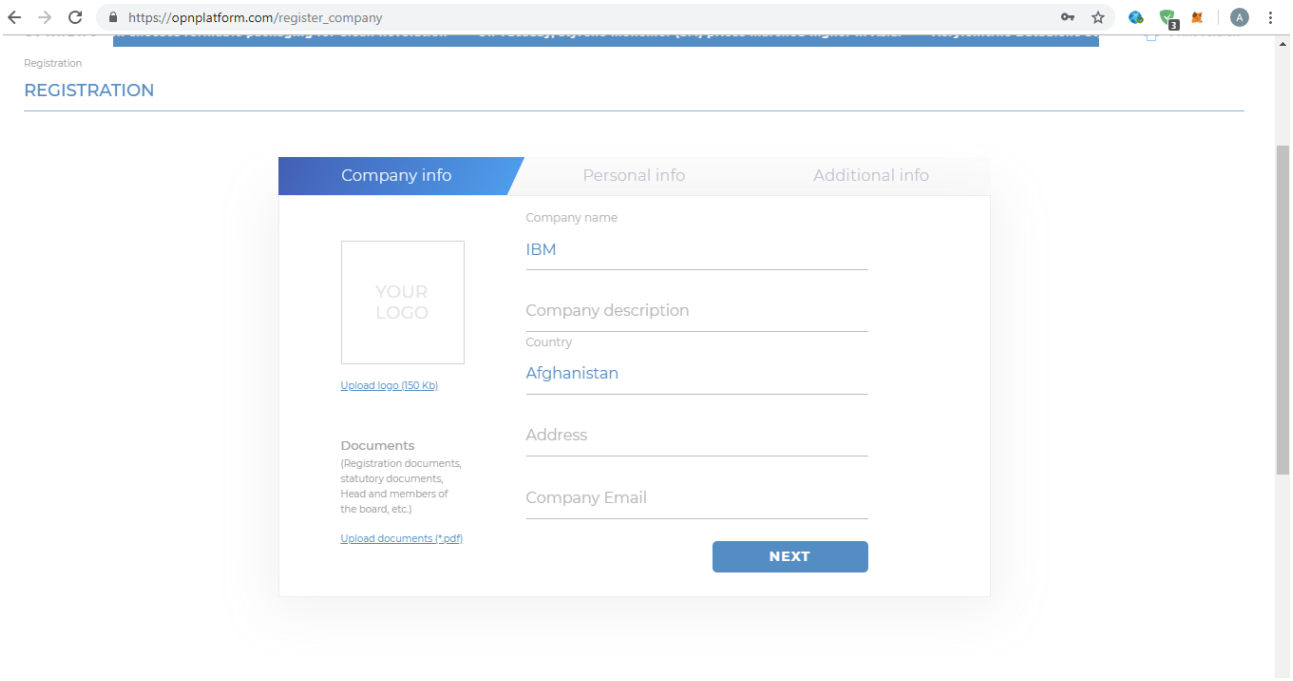
Усі паролі користувачів повинні зберігатися у хешованому вигляді, адже це допомогає запобігти витоку даних.



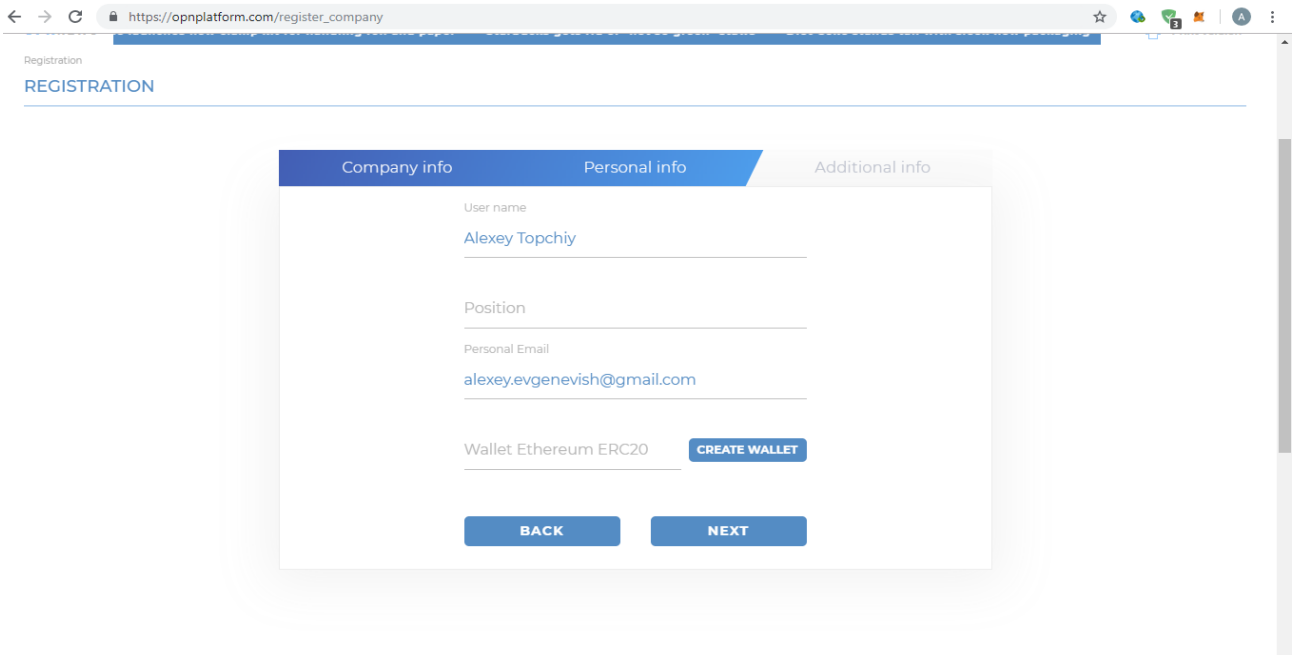
Зображення сторінки для відновлення паролю

3.3 Заповнення й маніпуляції з профілем компанії

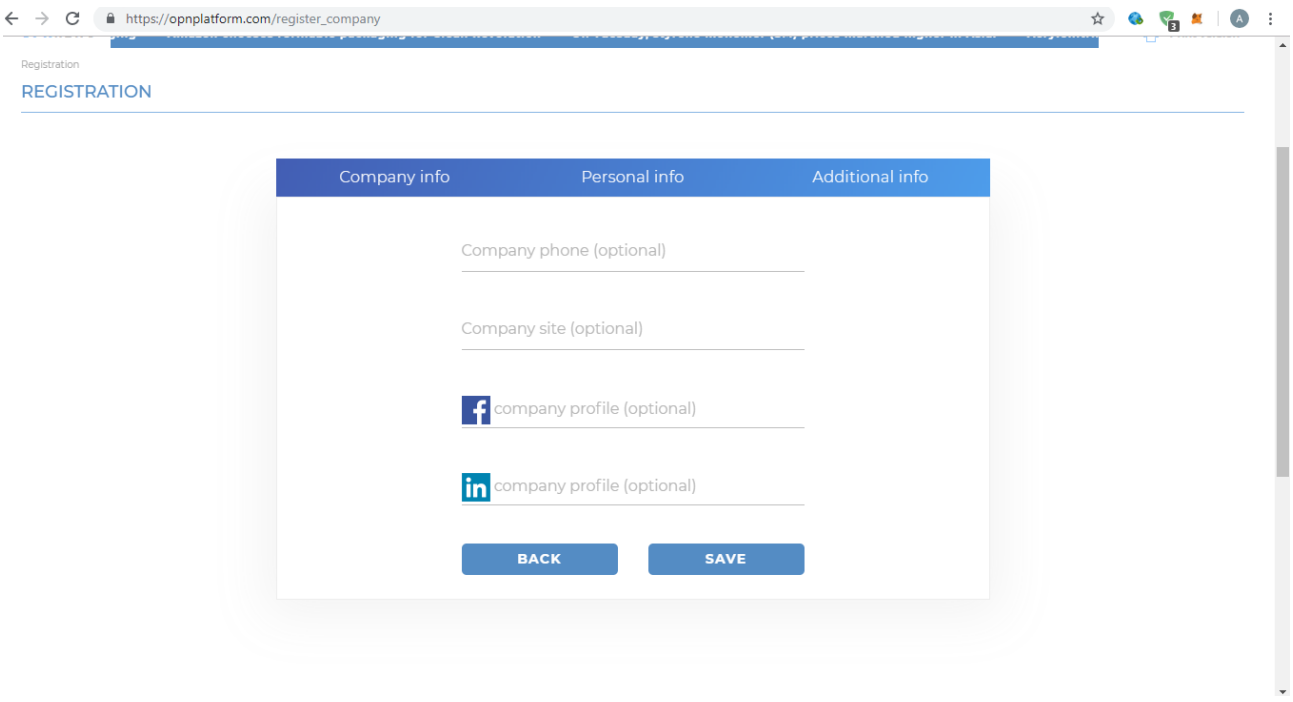
По завершенню реєстрації платформа повинна запросити в користувача інші дані про профіль компанії. Тому, одразу після першої авторизації, користувача зустрічає така трьох-рівнева форма:



Рівень 1



Рівень 2



Рівень 3

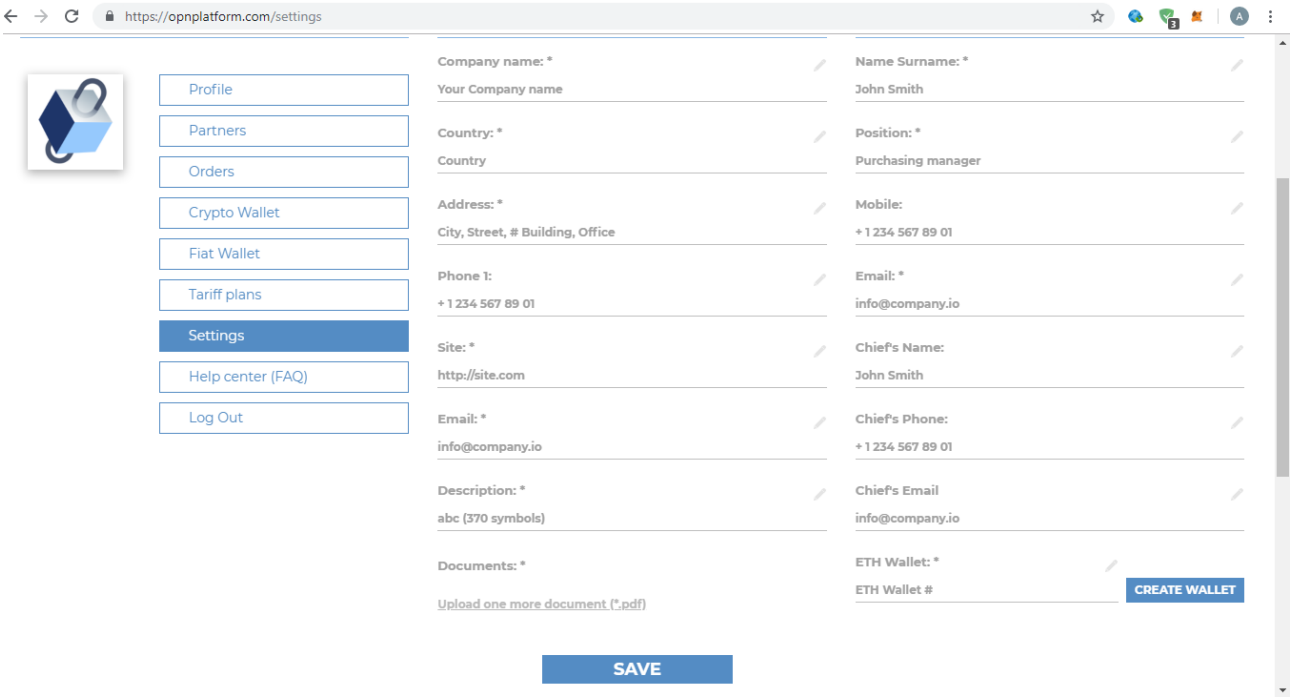
Ця форма збирає наступну інформацію про компанію-користувача:

1. Назва;
2. Логотип;
3. Декілька документів про діяльність компанії;
4. Країна, де працює компанія, або країна де розташований головний офіс;
5. Опис діяльності компанії;
6. Адреса офісу компанії;
7. Адреса корпоративної електронної скриньки компанії;
8. Повне ім’я користувача, який реєструє профіль компанії;
9. Позиція користувача у компанії, яку він реєструє;
10. Персональна електронна адреса користувача;
11. Адреса Ethereum гаманця компанії;
12. Телефон компанії (опціонально);
13. Посилання на веб-сайт компанії (опціонально);
14. Посилання на корпоративну сторінку у Facebook (опціонально);
15. Посилання на корпоративну сторінку yLinkedIn (опціонально).

Цю форму можна легко пропустити й продовжити використання платформи,

але вже у режимі «знайомства», у якому можливості користувача будуть обмежені. Наприклад, він зможе досліджувати товари, які продають та шукають на платформі інші користувачі, але сам не зможе робити того ж самого, доки не заповнить усі необхідні поля профілю своєї компанії.

Але користувач завжди зможе змінити або заповнити необхідні дані у розділі «Settings» у головному меню свого облікового запису. Там його зустріне наступна форма:

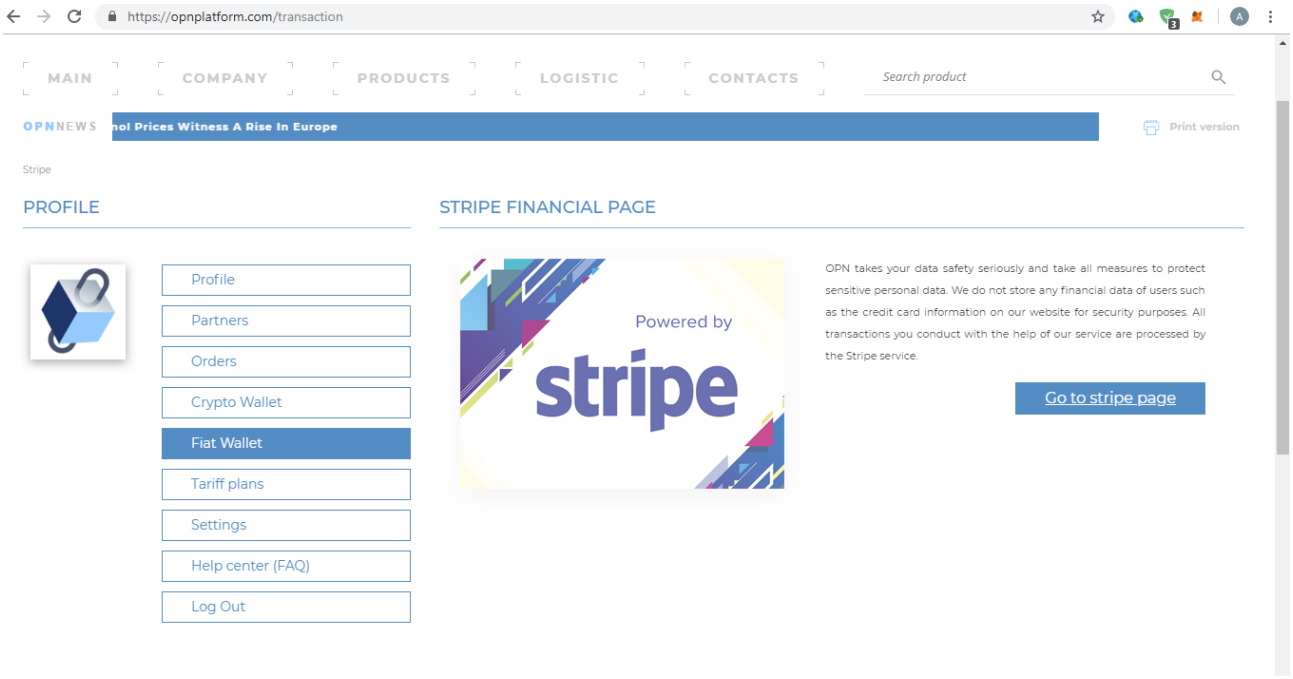


Форма для редагування інформації профіля компанії

Як тільки користувач заповнить усі необхідні поля профілю своєї компанії, останнім кроком для нього залишиться підключення інструменту для проведення транзакцій у фіатній валюті – облікового запису Stripe. Цей крок є також необхідним, адже він потрібен для надання широкого спектру можливостей до оплати та приймання коштів, коли компанія-користувач починає брати участь у торгівлі на платформі.

Для того, щоб підключити цей інструмент, треба мати вже створений обліковий запис Stripe й скористатись клавішою, яка переводить користувача на сторінку сайту Stripe для підключення аккаунтів клієнтів платформи до її власного аккаунту. Ця клавіша знаходиться у головному меню користувача у

пункті «Fiat wallet».



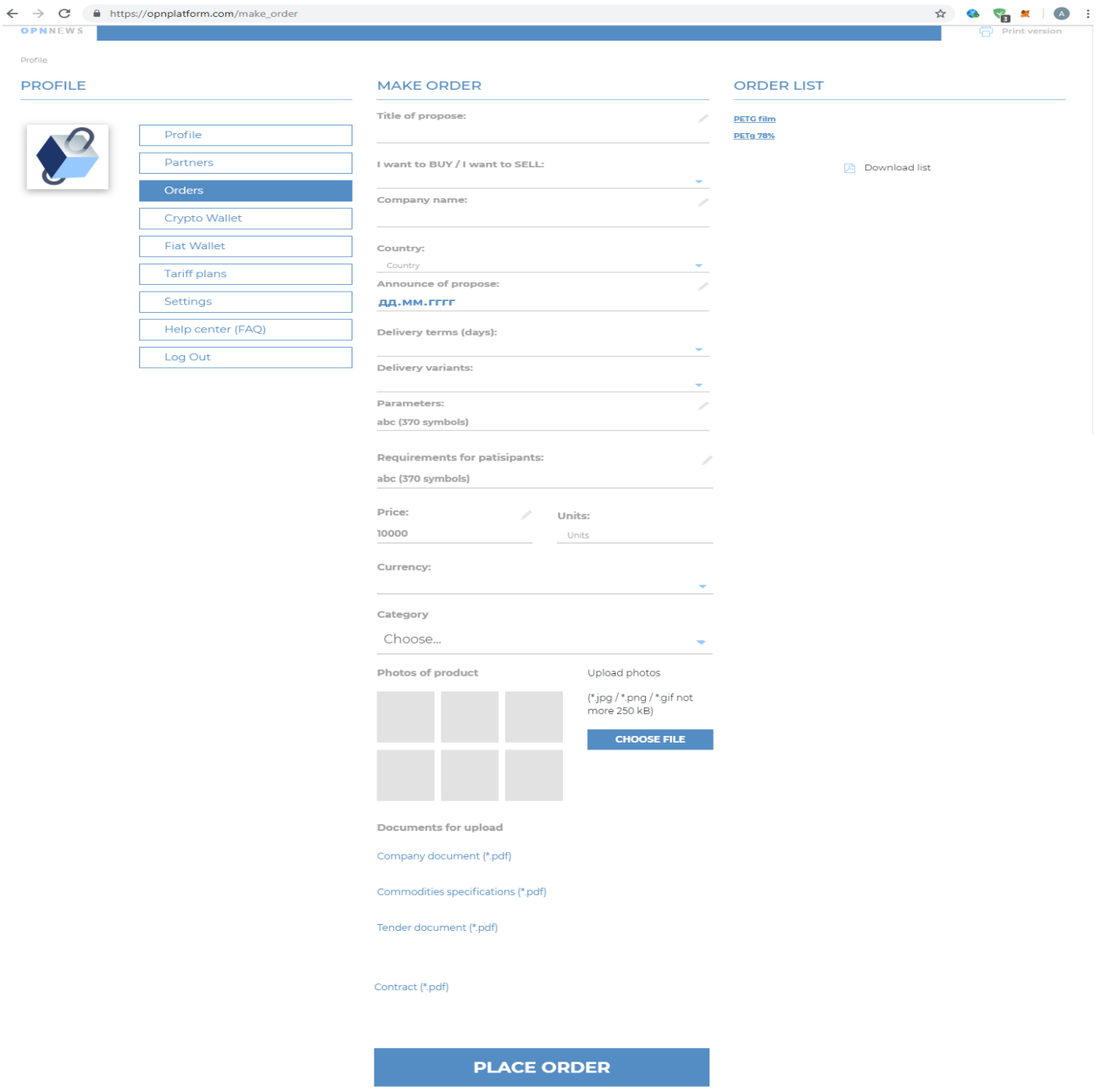
Після того, як усі ці кроки буде виконано, компанія зможе випустити свій перший лот на платформі, отримати перший прибуток й завести нових партнерів.

3.4 Створення лотів

У режимі «знайомства» користувач має можливість розробляти нові лоти, але не має можливості публікувати їх. Замість цього вони автоматично зберігаються до розділу «чернеток». Після заповнення усіх, необхідних для початку торгівельної діяльності, полів, компанія-користувач отримає можливість повноцінно взяти участь у торгівлі на платформі й публікувати свої власні лоти – як нові, так з розділу «чернеток». Для розробки нового лоту використовується форма з головного меню користувача у пункті «Orders». Ця форма збирає наступні дані:

1. Назва лоту;
2. Тип пропозиції: продаж або купівля;
3. Назва компанії, яка продає;
4. Країна з якої буде доставлятися товар;

5. Термін доставки;
6. Вид міжнародної доставки;
7. Параметри товару;
8. Вимоги до клієнтів;
9. Ціна за певну кількість;
10. Назва валюти, у якій буде укладена угода;
11. Категорія товару;
12. Фотографії товару;
13. Документи.

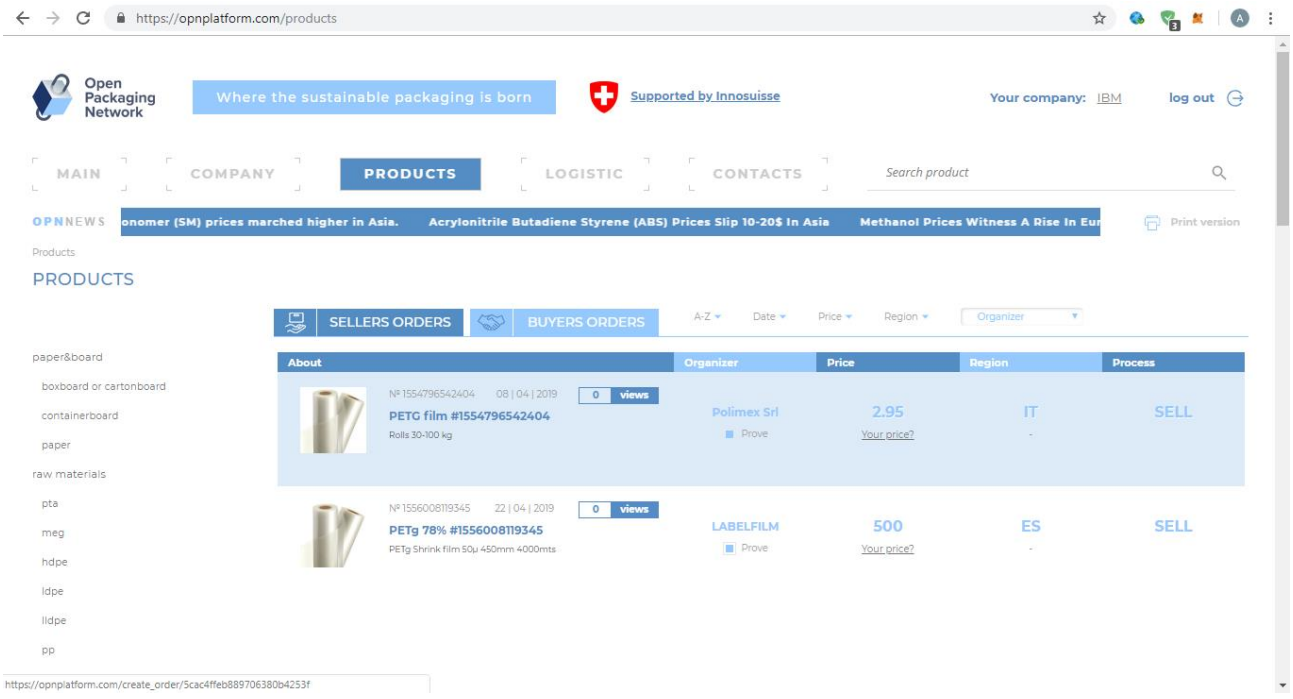


Повне зображення форми для складання нового лоту з головного меню

Коли користувачу буде потрібно створити лот, він зможе скористатись цією формою, або зробити витяг одної з чернеток лотів із бази даних й опублікувати її.

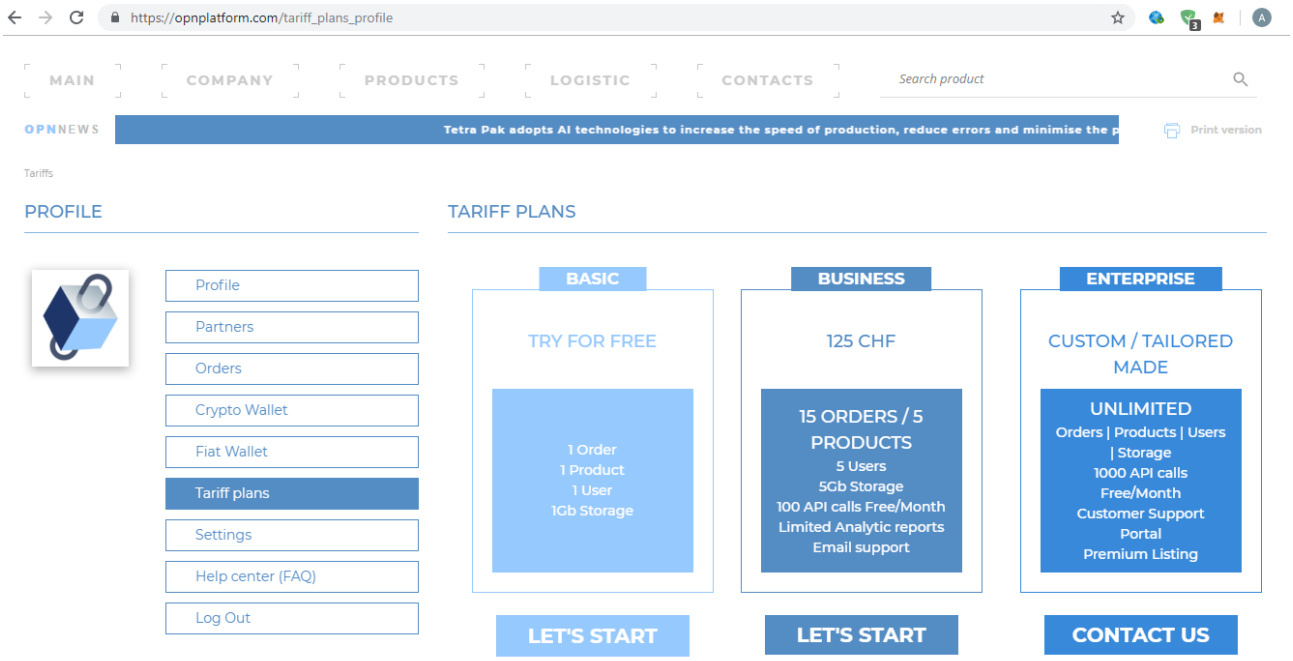
При публікації, кожен лот повинен в першу чергу пройти модерацію. Тому, без дозволу адміністратора, лот не буде видний іншим користувачам. Адміністратор у свою чергу контролює лоти через адмін-панель.

Для того, щоб отримати повний лист лотів н платформі, користувач повинени перейти у розділ «Products». Там, за допомогою спеціального API він отримає візуальний список усіх лотів на платформі.



Зображення візуального листу лотів на платформі

До речі, максимальна кількість розміщених лотів для однієї компанії обмежена за її робочим тарифом на платформі. Дізнатися про ціни й подробиці тарифів можна, перейшовши до пункту «Tariff plans» з головного меню.



3.5 Види взаємодії з лотами

Усього є 3 види взаємодії з лотами на платформі. У цьому підрозділі вони будуть розглянуті поступово.

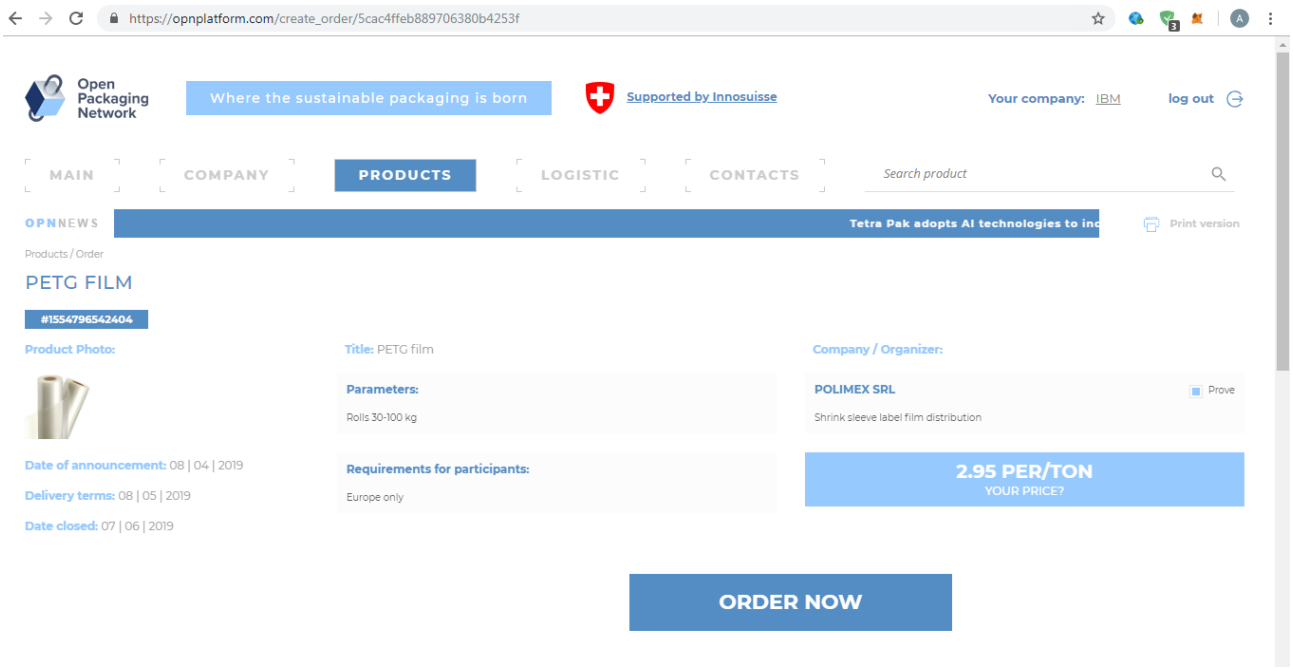
3.5.1 Замовлення

Замовлення – перший й найосновніший тип взаємодії з лотом на сайті торгівельної онлайн платформи, який полягає у купівлі або продажу товару за умовами, які вказані в інформації про лот.

Для замовлення ордеру, треба натиснути клавішу «Order now» на сторінці лоту. Тоді платформа запросить вид оплати: передоплата або оплата по факту доставки; й платіжні дані для використовування їх у Stripe.

Після цього платформа створить записи про угоду між двома компаніями й надасть їм обом тимчасовий чат для можливості спілкування один з одним на момент, доки очікується доставка товару. Очікується, що

надання чату для обох сторін посприяє зменшенню можливої недовіри між партнерами.

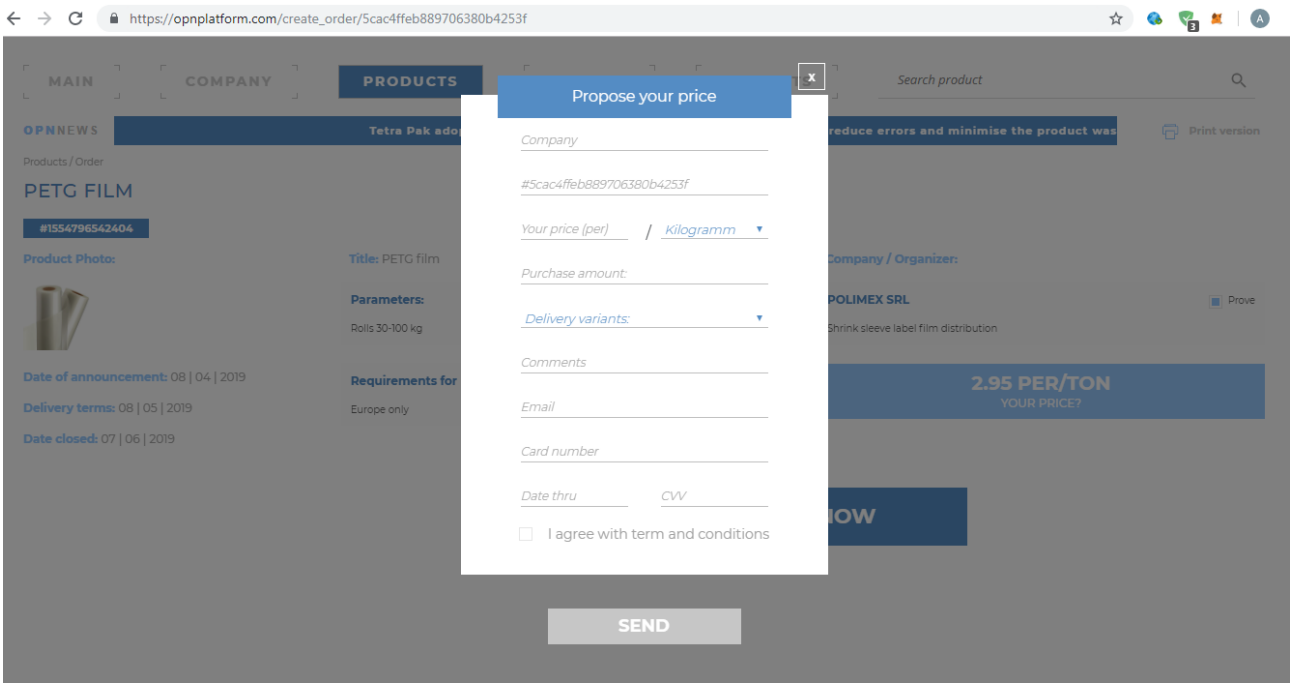


Зображення сторінки товару

3.5.2 Створення пропозиції до власника лоту

Як на деякому аукціоні або маркетплейсі, користувач отримає можливість запропонувати іншому користувачеві укласти угоду на його власних – першого користувача – умовах.

Власна пропозиція від користувача може відрізнятися від оригінальних умов лоту ціною, кількістю товару або типом доставки. Після того, як створюючий пропозицію користувач заповнює й відправляє форму, власнику лоту надходить повідомлення до електронної пошти про нову пропозицію. Після розглядання її умов, власник лоту має або відхилити пропозицію, або прийняти – тоді відкриється тимчасовий чат, як у разі звичайного замовлення.



Зображення форми для заповнення умов власної пропозиції

3.5.3 Створення рейтингу лоту

Рейтинг – невеликий, статистичний запис у базі даних про конкретний лот, який має кількість переглядів, загальне оцінювання та коментарі.

Кількість переглядів – загальне число запитів, яке інкрементується з кожним запитом інформації про конкретний лот від користувача, який не є його власником або співпрацівником його власної компанії.

Загальне оцінювання лоту – оцінка від 1 до 5 від кожного, хто заказав даний товар у даного продавця. Може бути поставлена 1 раз.

Коментар – невеличке повідомлення, яке буде бачити кожен, хто запитає інформацію про даний лот. Використовується, як засіб для надання відгуків про певний товар.

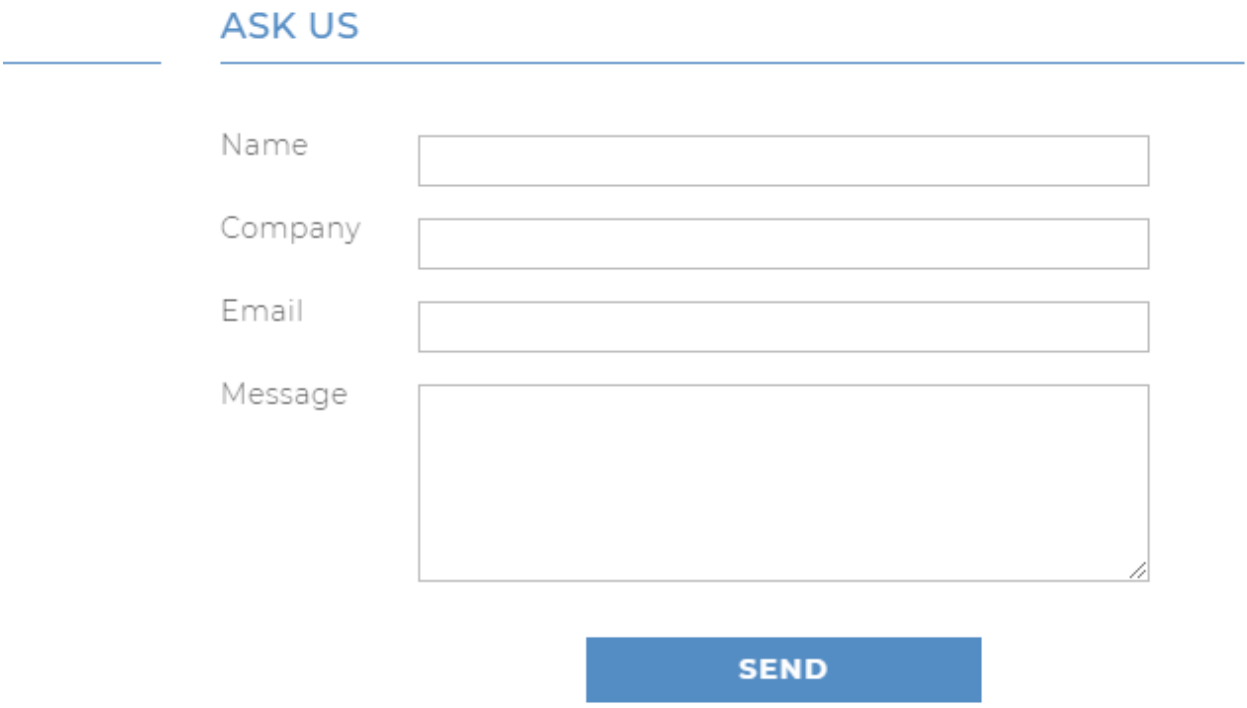
Ідентифікатор лоту – ідентифікатор запису про лот у базі даних онлайн платформи, який потрібний для додавання рейтингу до загального результату на запит інформації про цей сайт.

Частини, які складають повний запис про рейтинг лоту потрібні всі разом й кожна окремо, адже складання подібних рейтингів й оцінок допомагає краще, легше й швидше дослідити ринок й незабаром торгувати тільки тим, що має найвищий інтерес серед користувачів платформи.

3.6 Наявність зворотного зв’язку

На періоді адаптації до потреб користувача дуже важливо мати з ним якийсь діалог. Це допоможе краще зрозуміти його проблеми, бажання й пропозиції.

3.6.1 Форма зворотного зв’язку



Зображення форми, де кожний охочий може написати листа адміністрації

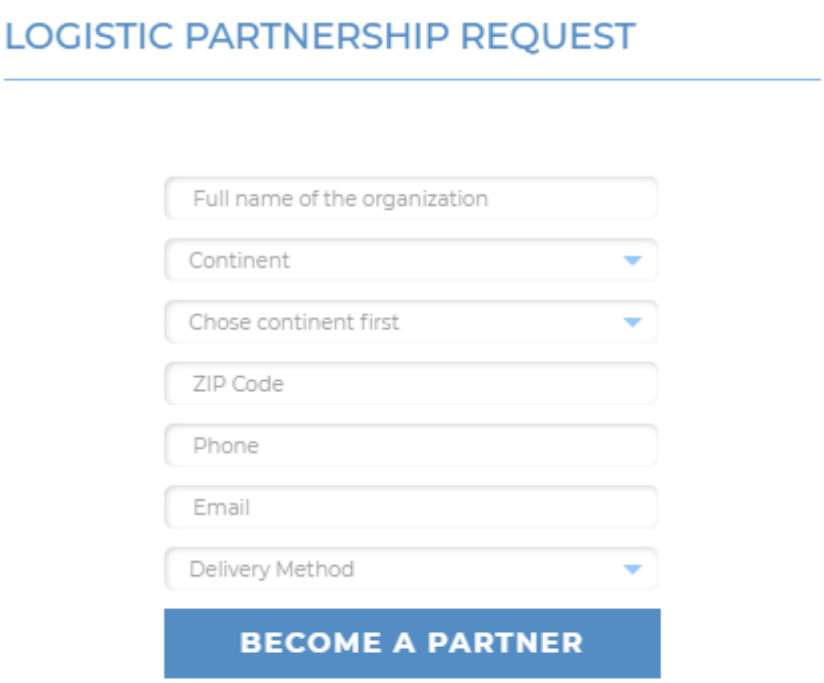
Така можливість може бути використана для надання повідомлення про несправність у системі, або якусь помилку. Ця форма також може бути формою для клієнтської підтримки, адже користувачу не має необхідності реєструватись, аби скористатися нею. Достатньо лише Вказати ім’я, назву компанії, адресу для відповіді й саме повідомлення. Адміністратор отримає цей лист до корпоративної електронної пошти.

3.6.2 Надання заявки на логістичне партнерство

Онлайн платформа для міжнародної торгівлі сировинними товарами не займається доставкою. Замість цього вона доручає доставку одному з логістичних партнерів платформи.

Логістичне партнерство є взаємовигідним, тому що платформа позиціонується, як майданчик для великої кількості людей, які торгують між собою, не зважаючи на різницю у відстані та територіальну проблему.

Обираючи логістичного партнера, треба точно знати де він знаходиться, тому процес подання заявки трішки більший, ніж процес відправлення звичайного відгуку.



Зображення форми для відправлення заявки для того, щоб стати партнером.

Для надійності вказаної географічної інформації, Країна обирається на основі обраного континента і вже потім веріфікується zip-code. Таким чином ми отримуємо точну адресу підприємства.

ВИСНОВКИ

1 Було проведено дослідження існуючих популярних торгівельних платформ. Під час цього дослідження було виявлено: чому вони є популярними; основні чинники, які посприяли їхньої успішності; основні переваги й недоліки; базові аспекти, які повинна мати торгівельна платформа.

2 У разі теоретичного планування платформи, можливих етапів його майбутнього розвитку, складання бізнес-моделі, виявлення цільових коирстувачів та розуміння їх основних проблем, було сплановано та розроблено основну інформаційну модель платформи.

3 На основі інформаційної моделі веб-сайту було сплановано та розроблено технічне завдання до функціоналу платформи й основні сценарії та принципи роботи користувачів із платформою. Розроблене технічне завдання було взято за основу при виборі технологій для реалізації функціоналу платформи. Таким чином було вирішено скористатися Express Framework, який надає зручний інтерфейс для розробки швидких та надійних веб-додатків на мові програмування Node.js . Ці дві технології були обрані, тому що є найбільш зручними для виконання цих задач. Для збереження даних була обрана СУБД MongoDB, тому що є бібліотека Mongoose, яка гарно адаптована під розробку на Node.js .

4 На підставі основних вимог до функціоналу платформи й можливих етапів її розвитку було вирішено використовувати архітектурний стиль «REST», створити єдиний API та використовувати його для роботи усіх клієнтів платформи.

5 Скориставшись результатами власних досліджень, спланувавши основне технічне завдання, розробивши інформаційну модель та спланувавши план дій було розроблено надійну, швидку та багатофункціональну платформу, яка буде надійним майданчиком для торгівлі між компаніями-користувачами та посприяє утворенню великого міжнародного, децентралізованого та незалежного ринку сировинних товарів.

СПИСОК ЛІТЕРАТУРИ

1. Ітан Браун - Веб-разработка с применением Node и Express;
2. Документація з Node.js;
3. Документація з MongoDB;
4. Документація з Mongoose;
5. “OAuth” Wikipedia;
6. “REST” Wikipedia;
7. “E-Commerce” Wikipedia;
8. Express.js Documentation;
9. “Ebay” Wikipedia;
10. Леонард Ричардсон та Михаель Амундсен - RESTful Web APIs: Services for a Changing World;
11. Subbu Allamaraju - RESTful Web Services Cookbook;
12. Express/Node introduction [Електронний ресурс] / Express/Node introduction. – Режим доступу: https://developer.mozilla.org/ru/docs/Learn/Server-side/Express_Nodejs/Introduction;
13. Нік Морган – «JavaScript для дітей: введення в програмування»;
14. Девід Херман - “68 конкретних шляхів використовувати JavaScript”;
15. Балабанов И.Т. Электронная коммерция. - СПб.: Изд. дом «Питер»: ЗАО «Питер бук», 2009;
16. Имери В. Как сделать бизнес в Internet: Пер. с англ. - 3-е изд. - К.; М.; СПб.: «Диалектика»;
17. Козье Д. Электронная коммерция. М.: ИТД "Русская редакция", 2009;
18. Тедеев А.А. «Электронная коммерция (электронная экономическая деятельность): правовое регулирование и налогообложение», 2009, «Приор»;

19. Закаблуцкая Е.В. Три проблемы электронной торговли// Управление магазином-2012;
20. Михайлова О. Проблемные вопросы развития электронной торговл-2012
21. Опарин А. Электронная коммерция на пороге 3D// E-Commerce World.-2012;
22.Салбер А. Перспективы развития рынка электронной коммерции/ Курьер печати- 2012.

ДОДАТОК

Ісходний код платформи

3.2.1.1

```
export const register = async (req,res) => {
   User.findOne({ 'mail.id': req.body.email }).exec(async (error,response) => {
      if(response) return helper.Error(res,400,'This email is already registered');
      let user;
      user = {
         name : req.body.name,
         mail: { id: req.body.email },
         pass: req.body.password,
         roles: 'USER',
         company_name : req.body.company_name,
         country : req.body.country,
         provedByPartner : !req.body.provedByPartner ? false : req.body.provedByPartner
      }

      let userObj = new User(user);
      userObj.save(async (error,response) => {
         if(error) throw helper.Error(res,400,error);
         let box = await messageBox.create({
            name : `${createBoxName(req.body.name)}.${createBoxName(req.body.company_name)}`,
            owner : { user : response._id }
         });
         let company_contact, contact = await hubspot.createOrUpdateContact(userObj.mail.id, {
            "firstname" : userObj.name.split(/\s/)[0],
            "lastname" : userObj.name.split(/\s/)[1]
         });
         contact = contact.body.vid;
         let email_domain = userObj.mail.id.split("@")[1];
         if ( free_email_domains.indexOf(email_domain) == -1 ) {
            let id = await hubspot.CompanyIdByDomain(email_domain);
            if ( id ) company_contact = id instanceof Array ? id[0] : id;
            else company_contact = await hubspot.createCompany(email_domain);
         }
         new Token(_.assign({
            uid : response._id,
            cid : process.env.WEB_CLIENT_CID,
            clientSecret : process.env.WEB_CLIENT_CLIENT_SECRET,
            token : crypt.serviceToken(),
            expires : Date.now()+7*24*60*60*1000,
            type : "CONFIRM_EMAIL"
         },{})).save(async (err, result) => {
            if (err) return helper.Error(res,400,err);
            let company ;
            try {
               company = await Company.create({
                  profile : {
                     name : userObj.company_name,
                  },
                  userInfo : {
                     employees : [
                        userObj._id
                     ],
```

```
                    admin : userObj._id
                  },
                  hubid : company_contact || null
                });
            } catch ( err ) {
              console.error( err );
              return helper.Error(res, 500, "An error occurred");
            }
            userObj.contact_id = contact;
            userObj.company_profile = company._id;
            userObj.mail.token = result._id;
            userObj.messageBox = box._id;
            files.checkUserFileDirs(userObj._id);
            userObj.save();
            email.transport.sendMail({
              from : email_config.smtp_config.auth.user,
              to : userObj.mail.id,
              subject : "Confirmation letter",
              html : email.welcomeLetterHTML(userObj.name, {
                logo_link : `${req.protocol}://${req.get('Host')}/logo/welcome.png`,
                confirm_link :
`${req.protocol}://${req.get('Host')}/api/v1/user/confirm_email/${result.token}`,
                terms_link : `${req.protocol}://${req.get('Host')}/public/terms.pdf`,
                privacy_link : `${req.protocol}://${req.get('Host')}/public/privacy.pdf`
              })
            }).then((response, error) => {
              if ( error ) return helper.Resp(res,400,error);
              userObj.password = "HIDDEN";
              userObj.salt = "HIDDEN";
              return helper.Resp(res,200,userObj);
            });
          });
      })
  });

}

export const confirmEmail = async (req,res) => {
  let token_param = req.params.token;
  if ( !token_param ) return helper.Error(res, 404, "Not found");
  Token.findOne({ token : token_param, type : "CONFIRM_EMAIL" }, (err, doc) => {
    if ( err || !doc || doc.expires.getTime() < Date.now() ) return helper.Error(res, 404, "Not found");
    User.findById(doc.uid, async (err, obj) => {
      if ( err || !obj || obj.mail.verified === true || obj.mail.token.toString() != doc._id.toString() ) return
helper.Error(res, 404, "Not found");
      obj.mail.token = null;
      obj.mail.verified = true;
      let contact = await hubspot.createOrUpdateContact(obj.mail.id, {
        "firstname" : obj.name.split(/\s/)[0],
        "lastname" : obj.name.split(/\s/)[1]
      });
      obj.save((err, result) => {
        if ( err ) return helper.Error(res, 500, "An error occured. Try later.");
        doc.remove();
        return res.redirect(`${req.protocol}://${req.hostname}/login`);
      });
    });
```

```
});Ï
```

3.2.1.2

```
export const oauthCallback = async (req,res) => {
  function callback(req, res) {
    return new Promise((resolve, reject) => {
      console.log(req.account);
      User.findOne({ "mail.id" : req.account.mail.id }, async(err, user) => {
        if ( err ) {
          console.error(err);
          reject("An error occurred");
        }
        if ( user && ( user.provider == "website" || user.provider != req.account.provider ) ) {
          reject("User is already signed with another authorization type")
        }
        if ( !user ) {
          user = await User.create(req.account).then((result, err) => {
            if ( err ) {
              console.error(err);
              reject("An error occurred");
            }
            return result;
          });
          let box = await messageBox.create({
            name : user.name,
            owner : { user : user._id }
          }).then((result, err) => {
            if ( err ) {
              console.error(err);
              reject("An error occurred");
            }
            else return result;
          });
          user.messageBox = box._id;
          try {
            console.log(user.company_name)
            let company = await Company.create({
              profile : {
                name : user.company_name,
              },
              userInfo : {
                employees : [
                  user._id
                ],
                admin : user._id
              }
            });
            user.company_profile = company._id;
          } catch ( err ) {
            console.error( err );
            reject( "An error occurred" );
          }
          user.save();
        }
        let pr = [];
        try {
          if ( req.accessToken.value )
```

```
                pr.push(token[req.account.provider].saveAccessToken(user._id, req.accessToken.value,
req.accessToken.expires));
              // if ( req.refreshToken.value )
              //     pr.push(token[req.account.provider].saveRefreshToken(user._id, req.refreshToken.value,
req.refreshToken.expires));
            } catch ( err ) {
              console.error( err );
              reject( "An error occurred" );
            }
            Promise.all(pr).then(async(result, err) => {
              if ( err )
              {
                console.log(err);
                reject ( "An error occurred" );
              }
              else {
                let tokenz = await token.getAuthTokenzByClientName(res, user._id, "WEB_CLIENT");
                files.checkUserFileDirs(user._doc._id);
                let obj = Object.assign(tokenz, user._doc);
                resolve ( obj );
              }
            });
          });
      });
    }
    callback(req,res)
    .then(resolve => {
      if ( req.method == "POST" ) helper.Resp(res, 200, resolve);
      else
      {

res.redirect(`/login?access_token=${resolve.access_token.token}&refresh_token=${resolve.refresh_token.to
ken}`);
      }
    })
    .catch(reject => {
      if ( req.method == "POST") return helper.Error(res, 400, reject);
      else res.redirect(`/login?error=${reject.replace(/\s/g, "%20")}`);
    })
}
```

3.2.2

```
export const login = async (req,res) => {
    User.findOne({'mail.id': req.body.email}).exec(async (error,response) => {
      if(error || !response) return helper.Error(res,421,'Email or password is incorrect');
      if(response.mail.verified === false) return helper.Error(res, 403, "Email is not verified");
      if(response.provider != "website") return helper.Error(res, 400, "User is already signed with another
authorization type");
      try {
        if(!response.checkPassword(req.body.password)) return helper.Error(res,421,'Email or password is
incorrect');
      } catch ( err ) {
        return helper.Error(res, 400, "Something wrong with user authentification");
      }
      let cid = req.body.clientId, uid = response._id;
      let tokenz_obj = await tokenz(res, cid, uid);
```

```
        response.password = "HIDDEN";
        response.salt = "HIDDEN";
        return helper.Resp(res,200,Object.assign(tokenz_obj, response._doc));
    });
}



export const hasAccess = (req,res,next) => {
    let check = v.validate(req.body, { access_token : "string" } );
    if(check == true) {
        Token.findOne({ cid : req.body.clientId, token : req.body.access_token },(err, token) => {
            if ( err || !token ) return Error(res, 401, "Unauthorized");
            if ( token.expires < Date.now() ) return Error(res, 401, "Invalid Token Error");
            if ( !mongoose.Types.ObjectId.isValid(token.uid) ) return Error(res, 500, "Wrong uid");
            User.findById( token.uid, async (err, user) => {
                if ( err || !res ) return Error(res, 401, "Invalid Token Error");
                res.locals =  { token, user, cid : req.body.clientId };
                next();
            });
        });
    }else{
        return Error(res,400,check);
    }
}
```

3.2.3

```
export const resetPassword = (req,res) => {
    User.findOne({ "mail.id" : req.body.email }, (err, response) => {
        if ( err ) return helper.Error(res,400,"Invalid email");
        if ( !response ) return helper.Error(res,400,`User with email ${req.body.email} doesn't exist`);
        if ( response.provider != "website" ) return helper.Error(res,400,`User with email ${req.body.email}
doesn't exist`);
        if ( response ) {
            new Token(_.assign({
                uid : response._id,
                token : crypt.serviceToken(),
                expires : Date.now()+60*60*1000,
                type : "RESET_PASSWORD"
            },{})).save(async (err, result) => {
                if ( err ) return helper.Error(res,400,err);

                const approve_url = `${req.protocol}://${req.get('Host')}/reset_password/?${result.token}`;

                email.transport.sendMail({
                    from : email_config.smtp_config.auth.user,
                    to : response.mail.id,
                    subject : "Password reset",
                    text : approve_url
                }).then((response, error) => {
                    if ( error ) return helper.Resp(res,400,"Invalid email");
                    return helper.Resp(res,200,"Letter with recover link was sent");
                });
            });
        }
    });
}
```

```
export const resetPasswordAction = (req, res) => {
   let token = req.params.token;
   if ( !token ) return helper.Error(res, 400, "Reset token is required");
   Token.findOne({ token : token, type : "RESET_PASSWORD", expires : { $gt : Date.now() } }, (err,doc)
=> {
      if ( err || !doc ) return helper.Error(res, 400, "Invalid Token Error");
      User.findById(doc.uid, (err, user) => {
         if ( err || !user ) return helper.Error(user, "Invalid Token Error");
         user.pass = req.body.password;
         user.save().then((result,err) => {
            if ( err ) next("Error");
            Token.deleteOne(doc).exec();
            return helper.Resp(res, 200, "Reset Success");
         });
      });
   });
}
```

3.3

```
export const updateProfile = async (req, res) => {
   if ( !res.locals.company ) return helper.Error(res, 400, "Create company profile first");
   if ( res.locals.company.userInfo.admin.toString() != res.locals.user._id.toString() ) return helper.Error(res,
403, "You don't have permission to do that");
   if ( req.body.profile ) {
      if ( req.body.profile.name ) {
         try {
            let dup_name = await Company.findOne({
               "profile.name" : req.body.profile.name,
               "userInfo.admin" : { $ne : res.locals.user._id }
            }).exec();
            if ( dup_name )
               return helper.Error(res, 400, "The company with given name already exists. Please use another
one!");
            else {
               res.locals.company.profile.name = req.body.profile.name;
               res.locals.user.company_name = req.body.profile.name;
            }
         } catch ( err ) {
            console.error( err );
            return helper.Error(res, 500, "An error occurred");
         }
      }
      res.locals.company.profile.description = req.body.profile.description || res.locals.company.description;
      if ( req.body.profile.location ) {
         res.locals.company.profile.location.country = req.body.profile.location.country ||
res.locals.company.profile.location.country;
         res.locals.company.profile.location.address = req.body.profile.location.address ||
res.locals.company.profile.location.address;
         res.locals.company.profile.location.JurisdictionOfIncorporation =
req.body.profile.location.JurisdictionOfIncorporation ||
res.locals.company.profile.location.JurisdictionOfIncorporation;
         if (!res.locals.company.profile.location.JurisdictionOfIncorporation &&
            res.locals.company.profile.location.country && (
            res.locals.company.profile.location.country.toLowerCase() == "us" ||
            res.locals.company.profile.location.country.toLowerCase() == "usa" ||
```

```
            res.locals.company.profile.location.country.toLowerCase() == "united states" ||
            res.locals.company.profile.location.country.toLowerCase() == "ca" ||
            res.locals.company.profile.location.country.toLowerCase() == "canada" ))
            return helper.Resp(res, 400, "Jurisdiction Of Incorporation is required");
      }
      if ( req.body.profile.contacts ) {
         if ( req.body.profile.contacts.email ) {
            res.locals.company.profile.contacts.email = req.body.profile.contacts.email ||
res.locals.company.profile.contacts.email;
            let domain = res.locals.company.profile.contacts.email.split("@")[1];
            if ( free_email_domains.indexOf(domain) == -1) {
               let id = await hubspot.CompanyIdByDomain(domain);
               if ( !id ) {
                  let company = await hubspot.createCompany(domain);
                  res.locals.company.hubid = company;
               } else res.locals.company.hubid = id instanceof Array ? id[0] : id;
            }
         }
         res.locals.company.profile.contacts.site = req.body.profile.contacts.site ||
res.locals.company.profile.contacts.site;
         res.locals.company.profile.contacts.phone1 = req.body.profile.contacts.phone1 ||
res.locals.company.profile.contacts.phone1;
         // res.locals.company.profile.contacts.phone2 = req.body.profile.contacts.phone2 ||
res.locals.company.profile.contacts.phone2;
      }
   }
   if ( req.body.financialInfo ) {
      res.locals.company.wallet = req.body.financialInfo.wallet || res.locals.company.wallet;
   }
   if ( req.body.userInfo ) {
         res.locals.user.name = req.body.userInfo.name || res.locals.user.name
         res.locals.user.position = req.body.userInfo.position || res.locals.user.position

   }
   res.locals.company.save().then((result, err) => {
      if ( err || !result ) return helper.Error(res, 500, "Profile update failure");
      else {
         res.locals.user.save();
         return helper.Resp(res, 200, "Profile update success");
      }
   });
}
```

3.4

```
export const createOrder = (req, res, next) => {
   let body = req.body;
   let photos = [], docs = [];
   let business_limit = ( req.body.purpose == "SELL" ) ? 15 : 5;
   if (res.locals.category.hidden == true ||
      res.locals.category.children.length > 0 ||
      !res.locals.category.parent )
      return helper.Resp(res, 400, "Invalid order category was given");
   if ( res.locals.photos && res.locals.documents ) {
      res.locals.photos.map(val => photos.push(val._id));
      res.locals.documents.map(val => docs.push(val._id));
   } else {
```

```
        return helper.Error(res, 400, "At least one photo and one document are required");
    }
    let order_obj = {
        type : body.purpose == "SELL" ? "PRODUCT" : "ORDER",
        name : body.title,
        purpose : body.purpose,
        description : body.description || "",
        requirements : body.purpose == "SELL" ? body.requirements || "" : null,
        currency : body.purpose == "SELL" ? body.currency || "" : null,
        price : body.price*100 || null,
        amount : body.amount,
        category : body.product,
        delivery : body.delivery || null,
        auction : body.auction || null,
        country : body.country || "",
        status : ( res.locals.empty_company_profile ) ? "DRAFT" : "PLACED",
        user : res.locals.user._id,
        company : res.locals.company._id,
        photos : photos,
        documents : docs,
        index : Date.now().toString()
    };
    let limit = res.locals.company.tariff == "BUSINESS" ? business_limit : 1;
    if ( res.locals.company.tariff == "ENTERPRISE" ||
        (res.locals.company.financialInfo.orders.length < limit && body.purpose == "BUY") ||
        (res.locals.company.financialInfo.products.length < limit && body.purpose == "SELL")) {
        Order.create(order_obj, (err, result) => {
            if ( err || !result ) console.log(err)//return helper.Error(res, 500, "An error occurred");
            else {
                if ( body.purpose == "SELL" ) res.locals.company.financialInfo.products.push(result._id);
                else res.locals.company.financialInfo.orders.push(result._id);
                res.locals.company.save().then(async(resolve, reject) => {
                    if ( resolve ) {
                        let stats = await OrderStats.create({ order : result._id });
                        console.log(stats);
                        result.stats = stats._id;
                        result = await result.save();
                        console.log(result);
                        res.locals.result = result;
                        next();
                    }
                    else {
                        result.is_deleted = true;
                        result.save();
                        return helper.Error(res, 500, "An error occurred");
                    }
                });
            }
        });
    } else return helper.Error(res, 403, `You have spent your order limit: ${limit}. Buy another tariff if you
want more orders`);
}
```

3.5.1

```
export const buyProduct = async (req,res,next) => {
    if ( !res.locals.user || !res.locals.order || !res.locals.order.user.messageBox )
```

```
    return helper.Error(res, 400, "Bad request");
  let token = await stripe.tokens.create({
    card : req.body.card,
  });
  if ( req.body.pay == "now" ) {
    try {
      let fee = await PriceList.findOne({},'fee').exec();
      fee = res.locals.order.price * fee.fee["order_buy"];
      res.locals.transaction.fee = fee;
      res.locals.transaction.amount_all = res.locals.transaction.amount + fee;
      await stripe.charges.create({
        amount : res.locals.order.price,
        currency : res.locals.order.currency,
        description :
          `Customer: ${res.locals.user.name}.
          Company: ${res.locals.company.profile.name}.
          Seller: ${res.locals.order.user.name}.
          Seller Company: ${res.locals.company.profile.name}.
          Operation: buy product ${res.locals.order.name}.`,
        source : token.id,
        destination : res.locals.order.user.customAccountId,
        application_fee_amount : fee
      });
    } catch( err ) {
      return helper.StripeError(res, err);
    }
  }
  else {
    res.locals.transaction.from.token = token.id;
  }
  let transact = await Transaction.create(res.locals.transaction);
  res.locals.deal.transaction = transact._id;
  let chat_obj = {
    name : `Buy order ${res.locals.order.name}`,
    members : [
      {
        box : res.locals.user.messageBox,
        user : res.locals.user._id
      },
      {
        box : res.locals.order.user.messageBox,
        user : res.locals.order.user._id
      }
    ]
  }
  Promise.all([
    Deal.create(res.locals.deal),
    Chat.create(chat_obj)
  ]).then(async(result, err) => {
    if ( err || !result ) {
      console.log(err);
      return helper.Error(res, 500, "An error occurred");
    } else {
      let prms = [];
      console.log(result[0]);
      console.log(result[1]);
      chat_obj.members.map(val => {
```

```
            prms.push(messageBox.findById(val.box).exec());
        });
        let boxes = await Promise.all(prms);
        boxes.map(val => {
            val.chats.push(result[1]._id);
            val.save();
        });
        res.locals.chat = result[1];
        next();
      }
   });
}
```

3.5.2

```
export const createOffer = async (req,res,next) => {
   res.locals.order = await res.locals.order;
   if ( res.locals.order.company.toString() == res.locals.company._id.toString() )
      return helper.Error(res, 403, "You can't do offers to yourself");
   let offer = {};
   offer.order = req.body.order;
   offer.price = req.body.offer.price;
   offer.amount = req.body.offer.amount;
   offer.currency = req.body.offer.currency;
   offer.payment = req.body.offer.payment;
   offer.deliveryVariant = req.body.offer.deliveryVariant;
   offer.comments = req.body.offer.comments;
   offer.email = req.body.offer.email;
   offer.tokenPayment = req.body.offer.tokenPayment;
   offer.status = "OFFERED";
   offer.from = res.locals.company._id;
   offer.to = res.locals.order.company._id;
   offer.buyer = res.locals.order.company;
   let token;
   try {
      token = await stripe.tokens.create({ card : req.body.card });
   } catch (err) {
      return helper.StripeError(res, err);
   }
   offer.buyerToken = token.id;
   Offer.create(offer, (err, result) => {
      if ( err || !result ) {
         console.log(err);
         return helper.Error(res, 500, "An error occurred");
      } else {
         res.locals.offer = offer;
         next();
      }
   });
}
```

3.5.3

```
export const rateOrder = async (req, res, next) => {
   Rate.create({
      from : res.locals.company._id,
      for : {
```

```
            order : {
                id : res.locals.order._id,
                stat : res.locals.order.stats._id,
            }
        },
        rank : req.body.rank
    }, async(err, rank) => {
        if ( err ) {
            console.error( err );
            return helper.Error(res, 500, "An error occurred");
        } else {
            res.locals.order.stats.rating.voices++;
            res.locals.order.stats.rating.summ+=req.body.rank;
            res.locals.order.stats.rating.value =
                res.locals.order.stats.rating.summ
                    /
                res.locals.order.stats.rating.voices;
            res.locals.order.stats.save().then((result, err) => {
                if ( err ) {
                    console.error( err );
                    return helper.Error(res, 500, "An error occurred");
                } else return helper.Resp(res, 200, "You successfully ranked this item");
            });
        }
    });
}
```