

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра системного аналізу та обчислювальної математики
(повне найменування кафедри)

Пояснювальна записка

до дипломної роботи

Бакалавра
(ступінь вищої освіти)

на тему Розробка веб застосунка для безпечних комерційних операцій
(назва теми)

Виконав(ла): студент(ка) 4 курсу, групи КНТ-810сп
Спеціальності 124 – Системний аналіз
(код і найменування спеціальності)

Освітня програма (спеціалізація)

Основи системного аналізу

БУРКОВСЬКИЙ Є. О.
(ПРИЗВИЩЕ та ініціали)

Керівник РЯБЕНКО А. Є.
(ПРИЗВИЩЕ та ініціали)

Рецензент ЛОЗОВСЬКА Л.І.
(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет _____ комп'ютерних наук і технологій
Кафедра _____ Системного аналізу та обчислювальної математики
Ступінь вищої освіти _____ бакалавр
Спеціальність _____ 124-Системний аналіз
(код і найменування)
Освітня програма (спеціалізація) _____ Основи системного аналізу
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

В. о. завідувача кафедри _____
Еліна ТЕРЕЩЕНКО
«09» червня _____ 20 23 року _____

**З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТА**

Бурковського Євгена Олександровича

(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема проекту (роботи) Розробка веб застосунка для безпечних комерційних операцій

керівник проекту (роботи) д.ф.-м.н., проф., РЯБЕНКО Антон Євгенович
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «30» травня 2023 року № 121

2. Строк подання студентом проекту (роботи) «9» червня 2023 року

3. Вихідні дані до проекту (роботи) В першому розділі представлено проблематику, наведено графіки шахрайства в різних країнах, розкрито тему актуальності проекту, наведені перелік аналогів, дате пояснення термінів. В другому розділі описана моя роль в данному проекті, описані архітектурні підходи та розкрито тему вибору архітектури, описані технології які будуть використовуватись при створенні, розроблено сам застосунок та представлено його візуалізацію, дан висновок.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

6. Консультанти розділів проекту

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1	Рябенко Антон Євгенович, к.ф.-м.н., доц.	01 березня 2023	09 червня 2023
2	Рябенко Антон Євгенович, к.ф.-м.н., доц.	01 березня 2023	09 червня 2023
3	Рябенко Антон Євгенович, к.ф.-м.н., доц.	01 березня 2023	09 червня 2023
Нормоконтроль	Ширококорд Д.В., к.ф.-м.н., ст. викл.	12 червня 2023	

7. Дата видачі завдання «18» квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Сформулювати мету на основні завдання дипломної роботи	02.04.2023 – 10.04.2023	
2	Розробка застосунку для розв’язку задачі	10.04.2023 – 1.05.2023	
3	Оформлення пояснювальної записки	1.05.2023 – 08.06.2023	
4	Попередній захист дипломної роботи та отримання рецензій	06.06.2023	
5	Захист дипломної роботи	12.06.2023	

Студент(ка)

_____ Євген БУРКОВСЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Антон РЯБЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Дипломна робота: 50 с., 23 рис., 1 дод.

Об'єкт дослідження – електронна комерція.

Предмет дослідження – вплив шахрайства при транзакціях куплі-продажу.

Мета роботи – розроблення застосунку для безпечної електронної комерції.

В дипломній роботі проаналізована проблема шахрайства при виконанні операцій куплі-продажу, надана статистика по країнах. Розроблений бекенд для застосунка на мові програмування Node.js, описані архітектурні підходи. Надані скріншоти інтерфейсу.

ESCROW, ЕЛЕКТРОНА КОМЕРЦІЯ, АРХІТЕКТУРА DDD, МОВА
ПРОГРАМУВАННЯ NODE.JS

ЗМІСТ

Завдання.....	2
Реферат.....	4
Вступ.....	6
1. Теоретична частина.....	7
1.1 Проблематика.....	7
1.2 Escrow.....	10
2. Практична частина.....	12
2.1 Роль на проєкті.....	12
2.2 Архітектура проєкту.....	14
2.3 Технології, фреймворки, бібліотеки.....	17
2.4 Візуалізація інтерфейсу.....	24
2.4.1 Реєстрація.....	24
2.4.2 Створення замовлення.....	36
Висновок.....	45
Перелік посилань.....	46
Додаток А. Вихідний код.....	47

ВСТУП

В сфері електронної комерції, де бізнес зустрічається з технологіями, щоб створити безперебійні та зручні покупки для клієнтів у всьому світі, у сучасному цифровому ландшафті, що швидко розвивається, електронна комерція стала трансформаційною силою, яка революціонізувала спосіб купівлі та продажу продуктів і послуг. Від невеликих стартапів до транснаціональних корпорацій, компанії будь-якого розміру використовують можливості електронної комерції для охоплення ширшої аудиторії, оптимізації операцій і стимулювання безпрецедентного зростання.

Електронна комерція, охоплює широкий спектр онлайн-транзакцій, включаючи роздрібну торгівлю в Інтернеті, електронні платежі, онлайн-аукціони та цифровий маркетинг. Завдяки використанню веб-сайтів, мобільних додатків і різноманітних онлайн-платформ електронна комерція дозволяє компаніям спілкуватися з клієнтами в будь-який час і в будь-якому місці, долаючи географічні межі та часові обмеження. Лише кількома клацаннями миші споживачі можуть переглядати величезний каталог продуктів, порівнювати ціни, читати відгуки та робити безпечні покупки, не виходячи з дому.

Зростанню електронної комерції сприяло кілька ключових факторів. По-перше, широке впровадження підключення до Інтернету та поширення смартфонів створили базу споживачів, які розуміються на цифрових технологіях, які прагнуть зручності, швидкості та персоналізованого досвіду. По-друге, прогрес у логістиці та управлінні ланцюгом поставок зробив можливим доставляти товари та послуги швидше та надійніше, часто перевершуючи традиційні звичайні моделі. Нарешті, все більша поширеність захищених онлайн-платіжних систем і заходів із зміцнення довіри вселили впевненість у клієнтів, спонукаючи їх прийняти онлайн-покупки як безпроблемний і безпечний варіант.

1 ТЕОРЕТИЧНА ЧАСТИНА

1.1 Проблематика

Шахрайство в інтернеті широко розповсюджене, а саме - шахрайство в електронній комерції. За цим поняттям розуміються шахрайські дії, що здійснюються окремими особами або групами, які видають себе за законних торговців або продавців у онлайн-транзакціях. Їхні дії націлені на ввід в оману нічого не підозрюючих покупців і можуть призвести до фінансових втрат або крадіжки особистих даних. Наведу приклади таких шахрайських дій:

1. *Фальшиві інтернет-магазини*: шахраї створюють професійні веб-сайти, які імітують справжні онлайн-магазини. Вони пропонують товари за привабливими цінами, щоб залучити покупців. Однак після здійснення оплати покупець, або отримує підробку, або неякісний товар, або товар не доставляється взагалі;

2. *Фішинг*: шахраї надсилають фішингові електронні листи або повідомлення, які майстерно маскують під перевірених продавців, наприклад: банків або онлайн-магазинів. Мета проста – інформація «жертви», часто просять дані рахунку або номери кредитних карток під приводом вирішення проблеми або підтвердження покупки. Звичайно надання цієї інформації може призвести до крадіжки особистих даних або несанкціонованих операцій. Даний вид шахрайства не втрачає своєї актуальності, так як в його основі лежить те, що людина схильна помилятися;

3. *Шахрайство з переплатою*: шахраї знаходять продавців та починають діалог вдаючи що зацікавлені в придбанні товару. Після цього вони надсилають платіж або підроблюють чек, на суму, вищу за узгоджену ціну, після цього вимагають у продавця повернення різниці, часто посилаючись на помилку. Шахрай навмисно квапить продавця, щоб він не помітив підробки. Після повернення різниці, продавець розуміє що платіж був підроблений, а гроші вже у шахрая;

4. *Шахрайство з недоставкою*: досить проста, але дієва схема, продавці після отримання грошей за товар довго затягують з його відправкою, затримують її на невизначений термін, підроблюють номери відстеження товару, та зрештою уникають відправки і зникають з вашими грошима;

5. *Підробка*: часто можна зустріти історії коли після замовлення товару отримуєш не те що замовляв, починаючи від підробки товару – закінчуючи цеглою. До цього виду шахрайства також відносяться речі, які доставляють з явними дефектами, або зовсім в неробочому стані, але це не завжди вина продавця, це комплексна проблема яка може корелювати як з людською помилкою так і з поганими послугами delivery-компанії

За даними Visa – 3/4 від усіх випадків шахрайства та витоків інформації припадає на електронну комерцію.



Рисунок 1.1 – Графік кількості створених фішингових ресурсів у світі

За інформацією на 2022 рік – близько 65% американських компаній були атаковані успішно і втратили дані або гроші. Ще 83% повідомляють про розсилку фішингових електронних листів.

Із 81% людей, які користувалися шахрайськими сайтами, 50% сказали, що втратили гроші.

Шахраї зазвичай діють через електронні листи, тексти або телефонні дзвінки. Лише небагато людей, які стали жертвами шахрайства за допомогою одного з цих методів, повідомили про втрату грошей. Наприклад, 81% учасників опитування, які зіткнулися із шахрайством через веб-сайт, заявили, що вони взаємодіяли з сайтом, а 50% втратили гроші. Ці дані показують, що метод контакту має кореляцію між залученням та грошовими втратами. Із 39% респондентів, з якими шахрай зв'язався телефоном, лише 11% сказали, що втратили гроші. Також 42% респондентів заявили, що зв'язувалися з шахраєм електронною поштою, і лише 13% із цієї групи сказали, що втратили гроші. Із 91% учасників, які заявили, що зв'язалися із шахраєм через соціальні мережі, 53% заявили, що втратили гроші.

За статистикою, у 2022 році на території Одеської області зареєстровано 5469 злочинів за статтею «Шахрайство», скоєних у сфері інформаційно-телекомунікаційних технологій, з них розкрито 757. Звісно, Україна вважається досить прогресивною країною, тому цифри не такі великі, але для країн менш прогресивних – цей показник разюче зростає, за статистикою Африканських країн, обман в сфері продажу онлайн товарів рахується сотнями тисяч. У нас в країні розповсюджені інтернет магазини з модерацією, такі як OLX, TORG, MODDO тощо. Але в країнах Африки наразі немає аналогів, тому там часто можна побачити продаж товарів через соціальні мережі, такі як Instagram, Whatsapp, звісно ці соціальні мережі не були створені для такого роду діяльності, тому допомогти постраждалим вони не в змозі [1].

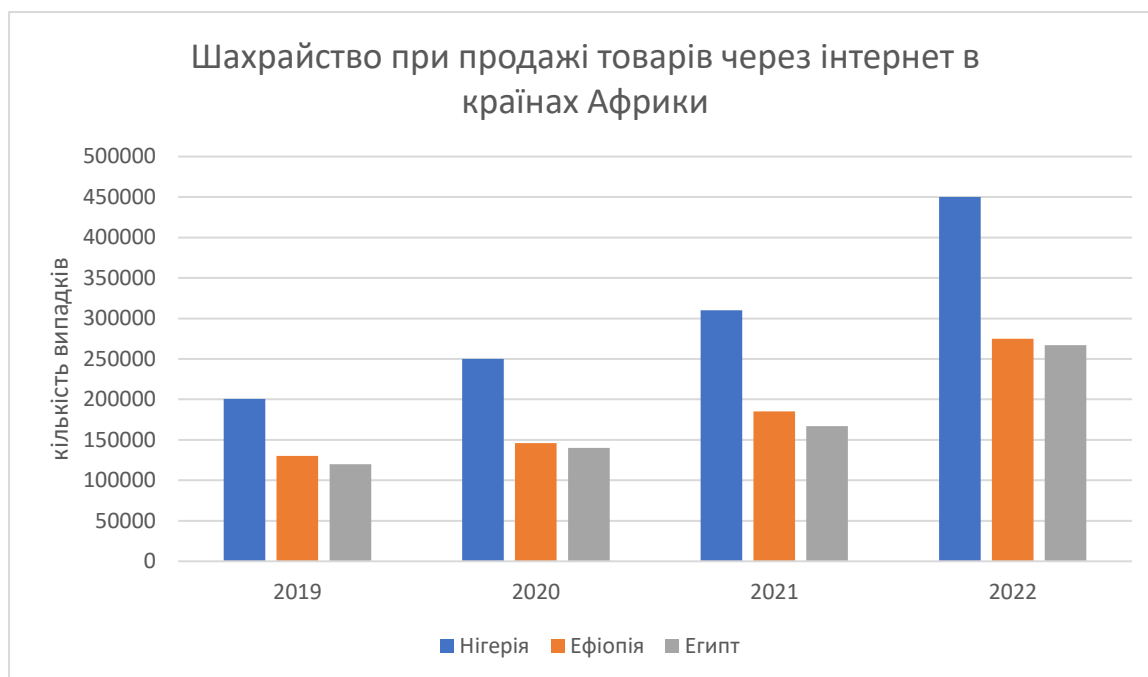


Рисунок 1.2 – Шахрайство при продажі товарів через інтернет в країнах Африки

З представленого вище графіка видно, що проблема шахрайства в сфері електронної комерції, а саме – транзакцій куплі-продажу товарів в інтернеті або соціальних мережах в країнах Африки не є маленькою, тому це можна розглядати як цільовий ринок для нашої програми, але потрібно розібратись чому так вийшло. З першого погляду це не є очевидним, бо в світі існує велика кількість застосунків з використанням Escrow підходу.

1.2 Escrow

Escrow або *депонування* – відноситься до фінансової або юридичної угоди, за якої неупереджена третя сторона володіє та управляє коштами або активами для кількох сторін, що беруть участь у правочині. Приклади включають рахунок, відкритий брокером для зберігання коштів від імені довірителя брокера чи іншої особи до завершення або припинення транзакції; або довірчий рахунок, який ведеться на ім'я позичальника для сплати зобов'язань, наприклад податків на

майно. Його метою є забезпечення безпеки та успішного виконання транзакції шляхом захисту коштів або активів доти, доки не будуть виконані всі заздалегідь визначені умови чи обов'язки.

Щоб було зрозуміліше, які застосунки використовують в своїй основі Escrow принцип, доцільно буде навести приклад найпопулярніших з них:

1. *Escrow.com*: є провідною онлайн-службою умовного депонування, яка спеціалізується на забезпеченні безпечних транзакцій для широкого кола галузей. Вона діє як третя довірена сторона між продавцем та покупцем, зберігаючи кошти на своїх рахунках. Цей застосунок підтримує різні типи транзакцій, починаючи від покупки автомобілю, закінчуючи продажем доменних імен та замовленням програмістів.

2. *PayPal Escrow*: хоча компанія PayPal більш відома за одноіменної платіжної системи, але також у них є даний застосунок який пропонує послуги умовного депонування. Це дозволяє продавцям і покупцям з додатковою безпекою проводити операції.

3. *Upwork*: одна з найпопулярніших фріланс бірж у світі, теж працює на принципах депонування. Допомогає мільйонам фрілансерів знайти роботу, а тисячам замовників – виконувачів. При виникненні суперечок, їх вирішує модератор. А гроші зберігаються на рахунку Upwork до вирішального рішення модератора.

4. *Bitrated*: це депонована платформа, спеціально розроблена для транзакцій із залученням криптовалют. Це дозволяє покупцям і продавцям брати участь у безпечних транзакціях із впевненістю, що кошти будуть вивільнені лише після виконання заздалегідь визначених умов.

5. *Freelancer.com*: Подібно до Upwork, Freelancer.com пропонує функцію депонування, відому як Milestone Payments. Це дозволяє роботодавцям створювати етапи та вносити кошти на умовне депонування, які передаються фрілансерам після досягнення етапів.

Milestone Payments – умовна назва так званих етапних платежів, теж цікавий підхід, який дозволяє більш детально розбити проект на етапи, узгодити їх терміни, обсяг роботи та оплату.

2 ПРАКТИЧНА ЧАСТИНА

2.1 Роль на проєкті

На цьому проєкті я відповідав за BackEnd та Devops. За професією я є Middle Nodejs developer. Наведу коротку інформацію про дані напрямлення, про їх зону відповідальності та роль у команді.

Бекенд-розробник — це інженер-програміст, який спеціалізується на створенні та підтримці веб-додатків або програмних систем на стороні сервера. Вони зосереджені на закулісній функціональності, яка забезпечує веб-сайт або програму, вирішуючи такі завдання, як керування базами даних, конфігурація сервера та розробка API.

Основний обов'язок розробника серверної частини полягає в тому, щоб серверні компоненти веб-програми або програмної системи функціонували ефективно та ефективно. Це передбачає роботу з такими мовами програмування, як Python, Java, Ruby або PHP, а також із такими фреймворками та технологіями, як Node.js, Django, Flask або Express.js.

Бекенд-розробники часто співпрацюють із інтерфейсними розробниками, які зосереджуються на користувацьких аспектах програми, щоб створювати цілісні та бездоганні системи. Вони об'єднують інтерфейсні та бекенд-компоненти, встановлюють зв'язок між різними частинами програми та реалізують необхідну бізнес-логіку та механізми обробки даних.

Крім того, бекенд-розробники несуть відповідальність за оптимізацію продуктивності, забезпечення заходів безпеки та впровадження рішень масштабованості для задоволення зростаючих баз користувачів або збільшених вимог до системи. Вони також можуть бути залучені до усунення несправностей і налагодження проблем, які виникають у серверних компонентах.

Таким чином, бекенд-розробник — це кваліфікований професіонал, який спеціалізується на створенні та підтримці серверної інфраструктури та функціональності веб-додатків або програмних систем, відіграючи вирішальну роль у наданні надійних і надійних програм кінцевим користувачам.

Розробник DevOps — це професіонал, який поєднує практику розробки програмного забезпечення з досвідом операцій, щоб полегшити співпрацю та оптимізувати життєвий цикл розробки та розгортання програмного забезпечення. Розробники DevOps працюють над інтеграцією команд розробки та операцій, інструментів і процесів, щоб забезпечити ефективне та надійне постачання програмного забезпечення.

Основна мета розробника DevOps — подолати розрив між командами розробки (Dev) і операційними (Ops) шляхом просування культури співпраці, автоматизації та постійного вдосконалення. Вони використовують різні інструменти та методології для автоматизації процесів, таких як створення коду, тестування, розгортання та моніторинг.

Розробники DevOps добре володіють мовами сценаріїв, інструментами керування конфігурацією та практиками інфраструктури як коду. Вони використовують такі інструменти, як Git, Jenkins, Docker, Kubernetes і різні хмарні платформи, щоб створювати відтворювані та масштабовані середовища розробки. Використовуючи принципи інфраструктури як коду, вони можуть автоматизувати надання ресурсів інфраструктури та керування ними.

Розробники DevOps також наголошують на безперервній інтеграції та безперервному розгортанні (CI/CD), що забезпечує часті та автоматизовані випуски програмного забезпечення. Вони впроваджують і підтримують надійні системи моніторингу та журналювання, щоб отримати уявлення про продуктивність, доступність і безпеку програм, забезпечуючи проактивну ідентифікацію та вирішення проблем.

Співпраця є ключовим аспектом ролі розробника DevOps. Вони тісно співпрацюють з розробниками, системними адміністраторами, групами забезпечення якості та іншими зацікавленими сторонами, щоб налагодити ефективні робочі процеси, реалізувати стратегії контролю версій і сприяти швидкому зворотному зв'язку. Вони використовують гнучкі методології та заохочують спілкування та спільну відповідальність між командами.

Таким чином, розробник DevOps — це кваліфікований професіонал, який поєднує досвід розробки програмного забезпечення зі знаннями операцій для

сприяння співпраці, автоматизації та постійного вдосконалення процесу розробки та розгортання програмного забезпечення. Їхня увага зосереджена на інтеграції практики розробки та експлуатації, використанні інструментів і методологій для підвищення продуктивності, надійності та масштабованості в розробці програмного забезпечення.

2.2 Архітектура проекту

Провівши аналіз, мною було прийнято рішення про використання на проекті архітектурного паттерну – DDD (Domain Driven Design)

Domain Driven Design (DDD) — це підхід до розробки програмного забезпечення, який наголошує на розумінні та моделюванні основної сфери діяльності для керівництва розробкою та впровадженням складних програмних систем. Він спрямований на узгодження програмного забезпечення з проблемною областю реального світу, забезпечуючи ефективне спілкування та співпрацю між експертами та розробниками.

В основі DDD лежить концепція «домену». Домен представляє конкретну область знань або проблемний простір, у якому працює програмна система. Він охоплює основні поняття, правила та процеси, які визначають бізнес або проблему, що вирішується.

У DDD основна увага зосереджена на глибокому розумінні домену через тісну співпрацю з експертами з домену. Розробники тісно співпрацюють із цими експертами, щоб визначити та охопити концепції домену, поведінку та зв'язки. Це розуміння потім перекладається на модель предметної області, яка служить спільною мовою та ментальною моделлю, спільною як для експертів у галузі, так і для розробників.

Модель домену служить основою програмної системи, охоплюючи основну бізнес-логіку та процеси. Він виражається через об'єкти, сутності, об'єкти значення, агрегати та служби домену, серед інших будівельних блоків. Ці

елементи створені для відображення реальних сутностей, зв'язків і поведінки домену.

DDD наголошує на важливості повсюдної мови, яка є спільною мовою, яку використовують як експерти домену, так і розробники для опису та обговорення домену. Усюдисуща мова допомагає усунути неоднозначність і забезпечує загальне розуміння концепцій і процесів домену протягом життєвого циклу розробки програмного забезпечення.

DDD також виступає за стратегічні шаблони проектування, такі як обмежені контексти та відображення контексту, щоб керувати складністю великомасштабних систем і уможливлювати модульні та зв'язані конструкції. Обмежені контексти визначають межі, в яких працює модель домену, забезпечуючи розділення та інкапсуляцію різних субдоменів. Відображення контексту має справу з інтеграцією кількох обмежених контекстів і встановлює зв'язок і співпрацю між ними.

Застосовуючи принципи та методи DDD, розробники можуть створювати системи програмного забезпечення, які тісно пов'язані з проблемною областю, заохочуючи підтримку, розширюваність та адаптивність. DDD заохочує ітераційну розробку, безперервне навчання та вдосконалення моделі домену в міру поглиблення розуміння предметної області.

Підсумовуючи, Domain-Driven Design (DDD) — це підхід до розробки програмного забезпечення, який надає пріоритет розумінню та моделюванню основної сфери діяльності. Він сприяє співпраці між експертами та розробниками предметної області, наголошує на створенні спільної мови та моделі домену, а також використовує шаблони стратегічного проектування для створення програмних систем, які точно представляють і вирішують складні проблеми реального світу [2].

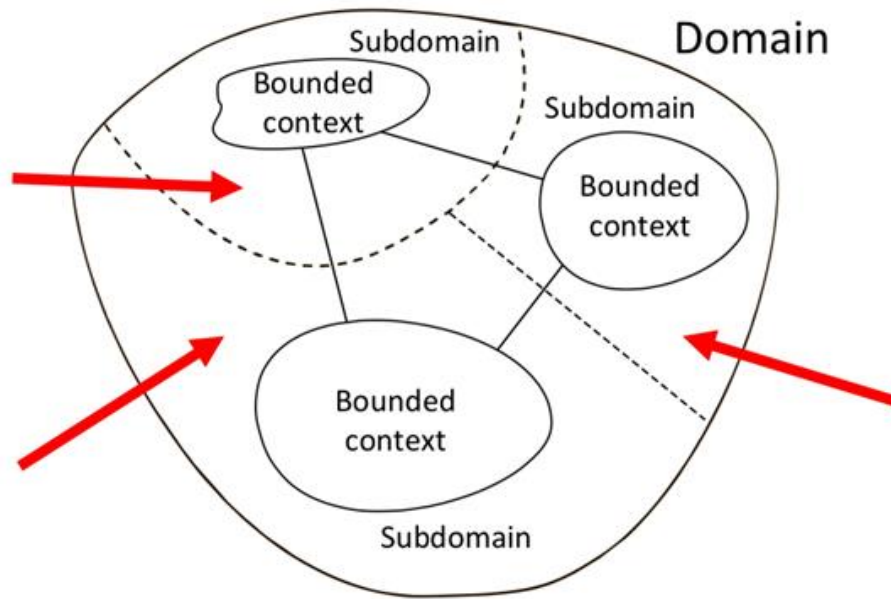


Рисунок 2.1 – Ілюстративне представлення домену

Використання архітектурних паттернів завжди зводиться до суперечок, тому що на реальному проєкті не очевидні границі доменів, та виникають суперечки при додаванні нових фіч, а саме до якого домену вона повинна входити, чи це вже новий домен у своєму початковому вигляді. Тому, щоб полегшити життя розробника, на дійсно великі проєкти часто додають окрему людину – архітектора, який займається питаннями архітектури застосунку, нажаль, при розробці даного застосунку я міг розраховувати лише на себе [3].

Для нашої програми я виділив такі основні домени як:

- Management – домен який відповідає за менеджрування користувачів, їх ролей, збереженням та логікою реєстрації та авторизації.
- Balance – домен відповідальний за збереження балансу користувача. Також до нього звертаються при запиті на зняття грошей.
- Order – одна з головних моделей, яка описує замовлення, логіку зміни статусів, збереження інформації щодо замовлення.
- Payment – доменна модель яка відповідає за створення транзакцій всередині застосунку, збереження платіжних даних користувача та зняття комісії.

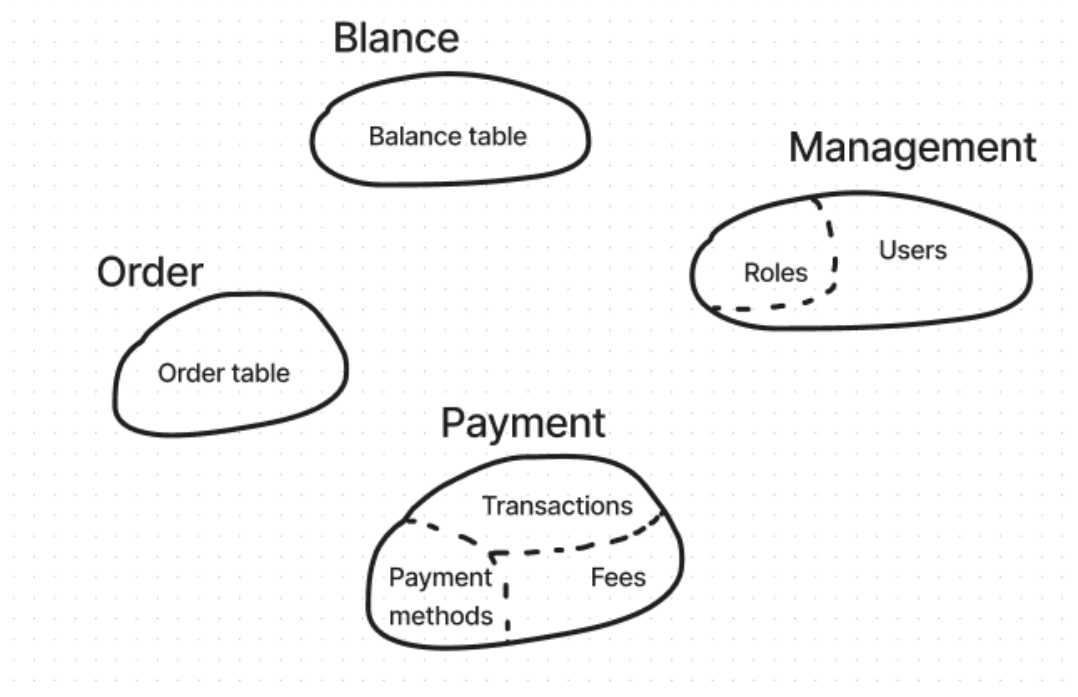


Рисунок 2.2 – Домени в застосунку

2.3 Технології, фреймворки, бібліотеки

JavaScript (JS) — це широко поширена мова програмування, яка в основному використовується для створення динамічних та інтерактивних елементів на веб-сторінках. Це інтерпретована мова високого рівня, яка дозволяє розробникам додавати функціональність, інтерактивність і поведінку до веб-сайтів.

JavaScript переважно виконується у веб-браузерах, що дозволяє йому маніпулювати елементами веб-сторінки, обробляти взаємодії користувачів, виконувати обчислення, перевіряти форми та динамічно оновлювати вміст без необхідності оновлення сторінки. Він надає потужний набір інструментів для створення адаптивного та привабливого веб-досвіду.

Основні функції JavaScript включають:

- **Об'єктно-орієнтоване програмування:** JavaScript підтримує принципи об'єктно-орієнтованого програмування (ООП), дозволяючи розробникам створювати та керувати об'єктами за допомогою властивостей і методів.
- **Програмування, кероване подіями:** JavaScript значною мірою покладається на події, які викликаються взаємодією користувача (наприклад, клацанням миші, введенням з клавіатури) або іншими діями (наприклад, завантаження сторінки, закінчення таймера). Розробники можуть реагувати на ці події та відповідним чином виконувати певний код.
- **Кросплатформна сумісність:** JavaScript підтримується майже всіма сучасними веб-браузерами, що робить його кросплатформною мовою, яка може працювати на різних операційних системах і пристроях.
- **Інтеграція з HTML і CSS:** JavaScript легко інтегрується з HTML і CSS, дозволяючи розробникам отримувати доступ і динамічно змінювати елементи веб-сторінки, стилі та атрибути.
- **Велика екосистема бібліотек і фреймворків:** JavaScript має величезну екосистему бібліотек і фреймворків (таких як React, Angular і Vue.js), які надають додаткові функції, полегшують веб-розробку та підвищують продуктивність.
- **Асинхронне програмування:** JavaScript підтримує шаблони асинхронного програмування, дозволяючи розробникам виконувати код одночасно, не блокуючи виконання інших завдань. Це особливо корисно для виконання трудомістких операцій або надсилення запитів до зовнішніх серверів.

JavaScript — це універсальна мова, яка виходить за межі веб-розробки. Він також використовується в серверних середовищах (наприклад, Node.js) і фреймворках розробки мобільних програм (наприклад, React Native, NativeScript) для створення повноцінних програм.

Node.js — це середовище виконання JavaScript на стороні сервера з відкритим кодом, яке дозволяє розробникам запускати код JavaScript поза веб-браузером. Це дозволяє виконувати JavaScript на сервері, відкриваючи можливості для створення масштабованих, високопродуктивних веб-додатків і служб [4].

Ключові функції та характеристики Node.js включають:

- Асинхронний і керований подіями: Node.js дотримується архітектури, керованої подіями, використовуючи неблокуючі операції вводу-виводу. Це дозволяє обробляти кілька запитів одночасно, що робить Node.js високоефективним і здатним обробляти велику кількість одночасних підключень.
- JavaScript на сервері: за допомогою Node.js розробники можуть використовувати JavaScript як на стороні клієнта (у веб-браузерах), так і на стороні сервера, забезпечуючи повну розробку JavaScript і повторне використання коду.
- Механізм JavaScript V8: Node.js побудовано на механізмі JavaScript V8, розробленому Google, який компілює код JavaScript у високооптимізований машинний код, що забезпечує чудову продуктивність.
- Легкий і масштабований: Node.js має полегшене середовище виконання та неблокуючу модель вводу-виводу, що робить його добре придатним для створення масштабованих програм у режимі реального часу. Він може ефективно обробляти велику кількість одночасних підключень, що робить його популярним вибором для додатків, які вимагають високої масштабованості та швидкості реагування.
- API на стороні сервера: Node.js надає багатий набір API для обробки мережевих протоколів, операцій файлової системи, потоків та інших функцій на стороні сервера. Ці API дозволяють розробникам створювати надійні та багатофункціональні серверні програми.
- Спільнота та екосистема: Node.js має яскраву та активну спільноту розробників, які сприяють його розвитку та надають підтримку. Екосистема збагачена різноманітними фреймворками, бібліотеками та інструментами, які розширюють можливості Node.js і спрощують процес розробки.

Також не треба забувати про фундаментальну особливість Node.js, а саме – так званий цикл подій. Він дозволяє ефективно обробляти одночасні операції. Він відповідає за керування та виконання асинхронних і неблокуючих операцій у програмах Node.js [4].

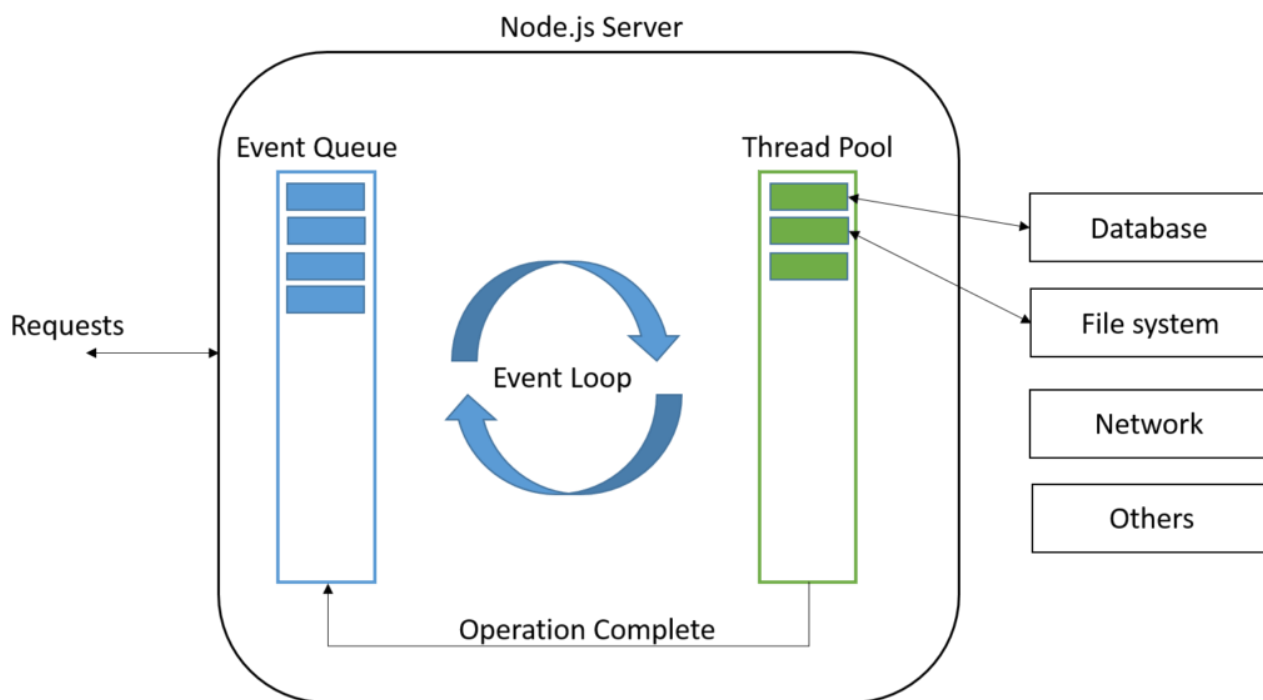


Рисунок 2.3 – Цикл подій в Node.js

Саме завдяки йому стає можливим обробка тисяч запитів до серверу без блокування та наявності лише 1 потоку. Фантастично просте та гарне рішення.

Вище зустрічалося поняття API, тому доцільно буде дати йому визначення. API (Application Programming Interface) – інтерфейс прикладного програмування. Це набір правил, протоколів і інструментів, які дозволяють різним програмам програмного забезпечення спілкуватися та взаємодіяти один з одним. API визначають методи та формати даних, які програми можуть використовувати для запиту й обміну інформацією, виконання дій і доступу до функцій іншої програмної системи, служби чи платформи.

API забезпечують стандартизований спосіб взаємодії різних програмних компонентів або систем, дозволяючи розробникам створювати програми, які можуть використовувати можливості інших систем без необхідності розуміти їхню внутрішню роботу. Вони діють як посередники, сприяючи спілкуванню та обміну даними між різними об'єктами програмного забезпечення.

API можна класифікувати на різні типи залежно від їх призначення та використання:

- Веб-інтерфейси API: ці інтерфейси API дозволяють програмам взаємодіяти з веб-службами або веб-платформами. Веб-інтерфейси API зазвичай відкриваються через протоколи HTTP(S) і використовують такі формати, як JSON або XML для обміну даними. Вони широко використовуються для створення веб-додатків, мобільних додатків та інтеграції зі сторонніми службами.

- API бібліотеки або фреймворку: бібліотеки та фреймворк надають набори попередньо створених функцій, класів і методів, які розробники можуть використовувати для спрощення завдань розробки. Ці API надають певну функціональність або абстракції для звичайних завдань, таких як обробка баз даних, інтерфейсів користувача або математичних операцій.

- API операційної системи: API операційної системи надають розробникам доступ до функцій і ресурсів операційної системи. Вони дозволяють програмам взаємодіяти з компонентами системи, такими як файлові системи, мережеві протоколи, апаратні пристрої тощо. API операційної системи відрізняються залежно від платформи, як-от Windows, macOS, Linux або мобільних операційних систем, як-от iOS і Android.

- Апаратні API: ці API надають доступ до апаратних функцій і функцій, таких як датчики, камери, GPS або інші спеціалізовані апаратні компоненти. Апаратні API дозволяють розробникам створювати програми, які використовують можливості базових пристроїв.

API зазвичай задокументовано з детальною інформацією про доступні кінцеві точки, методи, параметри, формати даних і механізми автентифікації. Розробники можуть інтегрувати API у свої програми, надсилаючи HTTP-запити, викликаючи функції бібліотеки або використовуючи конструкції, специфічні для мови програмування, для взаємодії з наданими інтерфейсами.

Використовуючи API, розробники можуть розширювати функціональні можливості своїх додатків, інтегруватися зі сторонніми службами та створювати модульні та сумісні програмні системи. API відіграють вирішальну роль у розробці сучасного програмного забезпечення, забезпечуючи безперебійну інтеграцію та співпрацю між різними програмними компонентами та службами.

Невід'ємно асоціюється з Node.js його пакетний менеджер – *NPM* (Node package manager), це менеджер пакетів за замовчуванням для Node.js, що надає величезну екосистему бібліотек і модулів із відкритим кодом, які розробники можуть використовувати для вдосконалення своїх програм. Це інструмент командного рядка, який спрощує процес керування та встановлення залежностей, спільного використання коду та керування робочими процесами проекту в проектах Node.js.

NPM дозволяє розробникам керувати залежностями проекту, вказуючи їх у файлі `package.json`. Цей файл містить список необхідних пакетів та їх версій. NPM вирішує та встановлює вказані пакети з реєстру NPM, який є централізованим сховищем загальнодоступних пакетів.

NPM надає команди для встановлення пакетів та їхніх залежностей із реєстру NPM. Команда `npm install` читає файл `package.json` і встановлює необхідні пакети локально в каталозі проекту. Він створює каталог `node_modules`, де зберігаються пакунки.

NPM дозволяє розробникам визначати конкретні версії або діапазони версій для залежностей. Це забезпечує послідовність і відтворюваність збірок. NPM надає команди для оновлення пакетів до їхніх останніх версій або встановлення певних версій відповідно до визначених критеріїв.

NPM дозволяє розробникам визначати власні сценарії у файлі `package.json`. Ці сценарії можна виконувати за допомогою команди `npm run`, що дозволяє автоматизувати типові завдання розробки, такі як створення, тестування та розгортання.

Розробники можуть публікувати власні пакети в реєстрі NPM, роблячи їх доступними для використання іншими. Це полегшує обмін кодом і співпрацю в спільноті Node.js.

NPM надає інструменти та функції для виявлення, спільного використання та спільної роботи над пакетами. Розробники можуть шукати пакети, досліджувати деталі пакетів і робити внески в проекти з відкритим кодом, розміщені в реєстрі NPM.

NPM підтримує концепцію пакетів з обмеженою областю, що дозволяє розробникам організовувати та розповсюджувати пакети в певному просторі імен. Пакунки з областю дії допомагають запобігти конфліктам імен і забезпечують логічну структуру для керування пов'язаними пакетами.

NPM став життєво важливим інструментом в екосистемі Node.js, що дає змогу розробникам використовувати величезну бібліотеку попередньо створених функцій і оптимізувати процеси розробки. Це спрощує керування залежностями, полегшує повторне використання коду та сприяє співпраці в спільноті Node.js [4].

Що до фреймворку який я буду використовувати – його назва NestJs. Він заснований на іншій дуже популярній бібліотеці – Express js, даний фреймворк дозволяє пришвидшити розробку в декілька разів, а зрозумілість коду виростає в декілька разів [5].

2.4 Візуалізація інтерфейсу

2.4.1 Реєстрація

Кожен застосунок так чи інакше починається з реєстрації, будь то KYC (Know Your Customer), email/password чи якийсь інший її вид. В нашому випадку буде використовуватись найбільш надійна, а саме – реєстрація за допомогою OTP коду

OTP (One-time password) розшифровується як «одноразовий пароль». Це механізм безпеки, який використовується для надання додаткового рівня автентифікації для онлайн-сервісів, облікових записів або транзакцій. Коди OTP – це тимчасові одноразові паролі, які генеруються та надсилаються на зареєстрований мобільний пристрій або адресу електронної пошти користувача.

Метою кодів OTP є запобігання несанкціонованому доступу до конфіденційної інформації або облікових записів, вимагаючи від користувачів

надати унікальний код на додаток до свого звичайного імені користувача та пароля. Ці коди зазвичай чутливі до часу та дійсні протягом короткого періоду часу, як правило, для одного сеансу входу або транзакції.

Kropay

Buy and sell anything with trust.

Enter your mobile number

 +380

Phone number

Рисунок 2.4 – Вікно логіну / реєстрації

Я довго думав, як же зробити процес реєстрації та авторизації найбільш пристосований для користувача, щоб він не витрачав багато часу на натискання кнопок та переходів на інші сторінки, часто можна побачити відокремленні сторінки для реєстрації та авторизації, але навіщо це робити, коли зі сторони коду це все легко відслідкувати та забезпечити максимальний комфорт користувачеві.

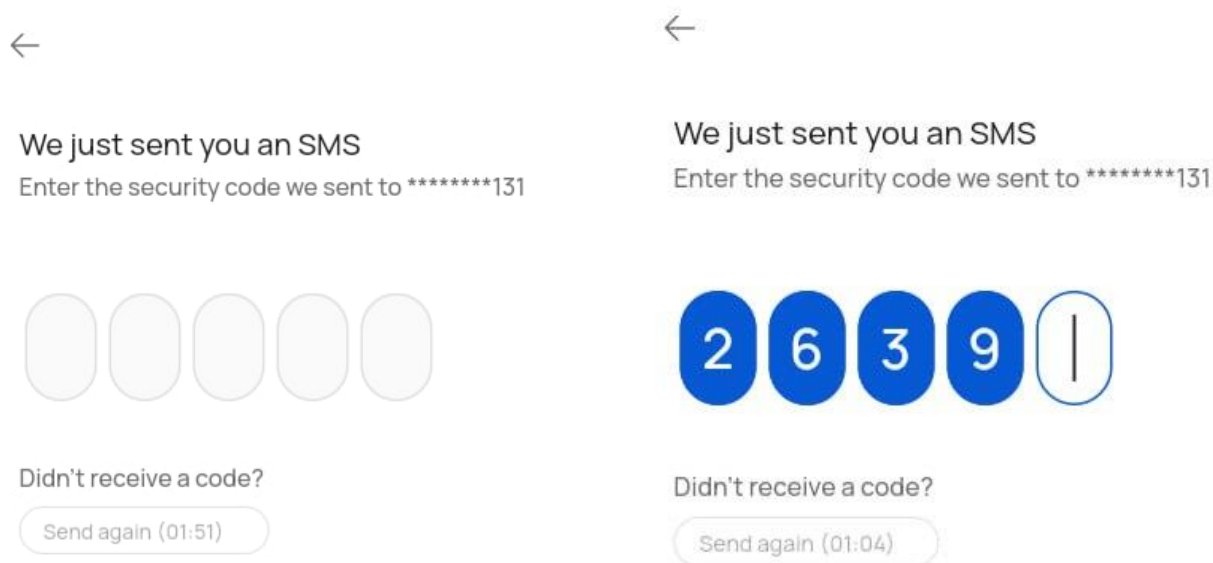


Рисунок 2.5 – Ввід OTP коду

Сам OTP приходить у вигляді СМС (Рис. 3.3), такий підхід дозволяє відмовитись від паролів, а тому – є найбільш захищений для користувача. Відправка СМС виконується за рахунок платного API – Twilio SMS.

Twilio SMS — це хмарна комунікаційна платформа, яка дозволяє компаніям надсилати й отримувати текстові повідомлення (SMS) програмним шляхом. Він надає набір API (інтерфейсів прикладного програмування), які розробники можуть використовувати для інтеграції можливостей SMS у свої програми, веб-сайти або системи.

За допомогою Twilio SMS розробники можуть легко включити можливість надсилати SMS-повідомлення на мобільні пристрої користувачів або отримувати вхідні SMS-повідомлення. Він пропонує просте та масштабоване рішення для компаній, щоб взаємодіяти зі своїми клієнтами за допомогою текстових повідомлень.

Загалом Twilio SMS спрощує процес інтеграції можливостей SMS у програми та системи, дозволяючи компаніям використовувати потужність текстових повідомлень для спілкування, взаємодії та підтримки клієнтів.

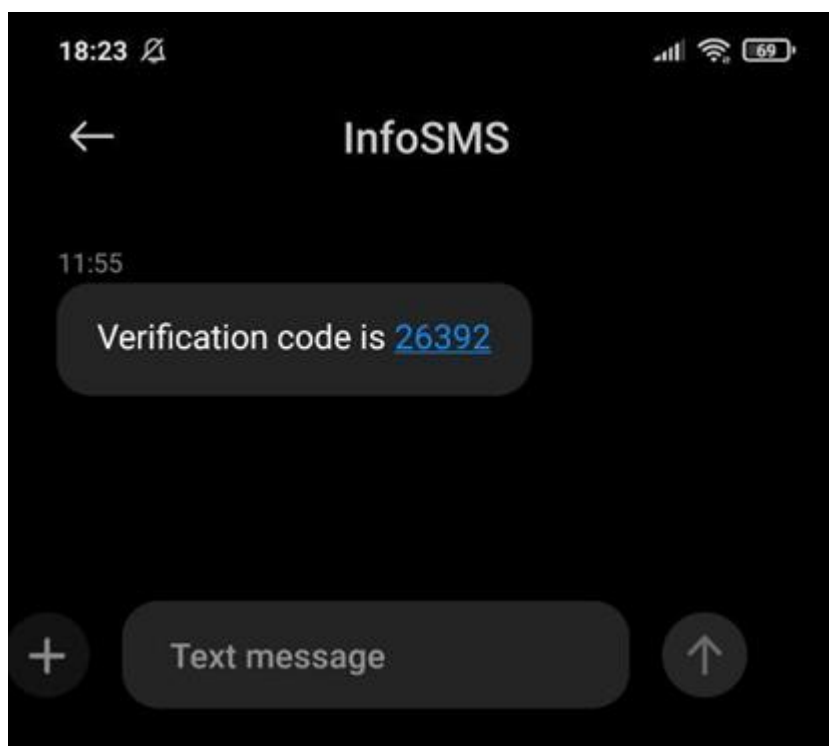


Рисунок 2.6 – Смс з ОТР кодом

Після цього, якщо користувач новий у нашій системі – йде багатоетапна реєстрація, яка складається з таких частин як:

1. Ввід email.
2. Підтвердження email шляхом вводу ОТР коду.
3. Ввід персональних даних.
4. Вибір як ви плануєте використовувати платформу.
5. Згода з T&C.



Enter your email address

Add your email to help make Kro secure
Verifying your email address helps you to
safely recover your password, retrieve and
protect your account.

Enter your email

example@gmail.com

Рисунок 2.7 – Ввід електронної пошти

Ця логіка дуже схожа з відправкою SMS, але для надсилання електронних листів використовую інший API – SendGrid.

SendGrid Email — це хмарна платформа доставки електронної пошти, яка надає підприємствам інструменти та інфраструктуру для надійного та масштабованого надсилання транзакційних і маркетингових електронних листів. Він пропонує набір API і служб, які розробники можуть використовувати для інтеграції функцій електронної пошти у свої програми чи системи.

SendGrid надає систему шаблонів, яка дозволяє компаніям створювати та керувати шаблонами електронної пошти для узгодженого брендування та обміну повідомленнями. Це спрощує процес створення електронних листів професійного вигляду з динамічним вмістом і персоналізованою інформацією.

SendGrid розроблено для обробки великих обсягів електронних листів, що робить його придатним для компаній будь-якого розміру. Незалежно від того, надсилаєте кілька сотень електронних листів або мільйони електронних листів на місяць, SendGrid може масштабуватися відповідно до потреб.



We just sent you an email

Enter the security code we sent to your email
some@gmail.com

Didn't receive a code?

Send again (01:48)

Рисунок 2.8 – Підтвердження електронної пошти

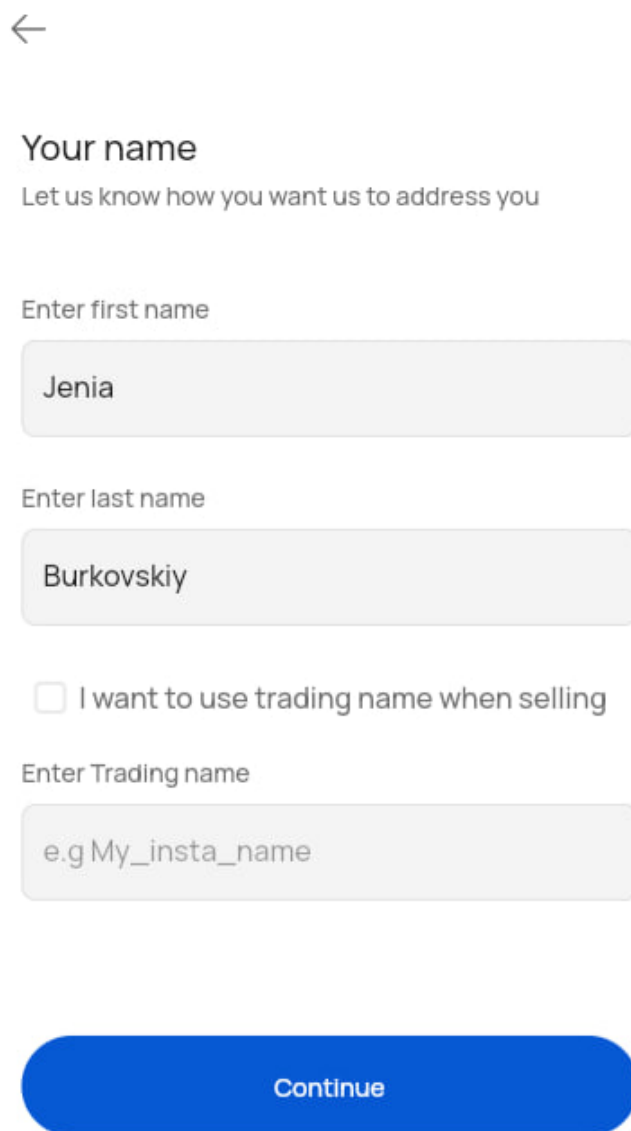
Далі, після підтвердження OTP коду – етап вводу персональних даних. Треба обговорити як генерується випадковий OTP код.

$$\text{MIN_CODE} + \text{Math.random()} * (\text{MAX_CODE} + 1 - \text{MIN_CODE})$$

Зрозуміло що MIN_CODE – мінімально можливий OTP код, а MAX_CODE – максимально можливий, тому наш випадковий код буде у проміжку між цими числами, але що ж за магічна функція Math.random()?

Math є статичним вбудованим об'єктом який є в JavaScript «з коробки», це вмістилище математичних констант та стандартних функцій.

Сам по собі метод random() повертає псевдовипадкове число від 0 до 1, яке я масштабую до потрібного мені діапазону.



←

Your name

Let us know how you want us to address you

Enter first name

Enter last name

☐ I want to use trading name when selling

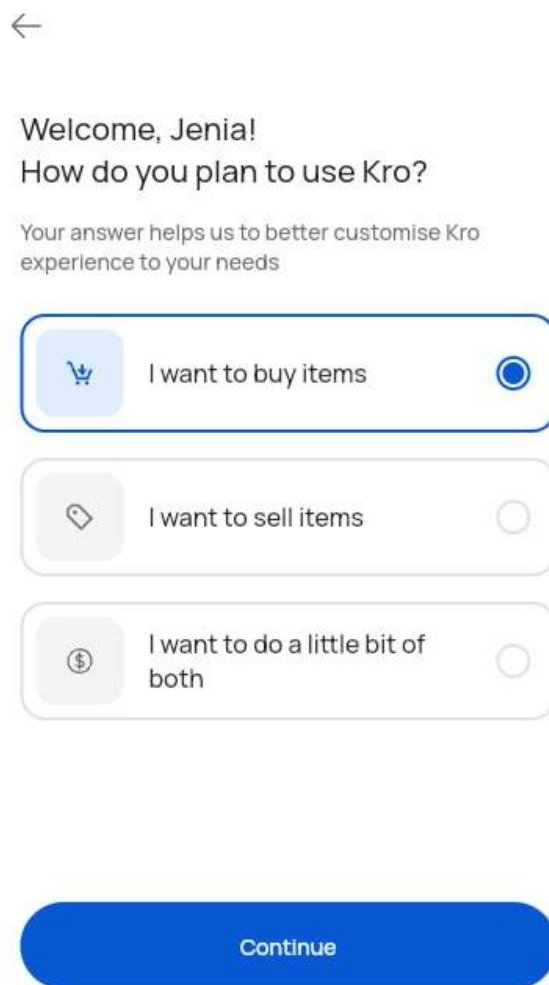
Enter Trading name

Continue

Рисунок 2.9 – Ввід персональних даних

Данна сторінка відповідає за надання користувачем його особистих даних, які в подальшому будуть використовуватись при транзакціях, та відображатись іншим користувачам. Trading name – повинно бути унікальним, для перевірки цього є відокремлений Endpoint (кінцева точка) та перевірка саме при доданні інформації, треба пояснити, що відокремлена кінцева точка існує лише для того щоб виводити користувачу помилку, говорячи що дане ім'я зайняте.

Кінцева точка – якщо говорити у контексті комп'ютерних мереж то це відноситься до певної URL або URI, тобто - розташування певної служби чи ресурсу до якого можна отримати доступ для маніпуляції, відправляючи HTTP запити.



←

Welcome, Jenia!

How do you plan to use Kro?

Your answer helps us to better customise Kro experience to your needs

☒ I want to buy items

☐ I want to sell items

☐ I want to do a little bit of both

Continue

Рисунок 2.10 – Вибір як ви плануєте використовувати платформу

Даний вибір суттєво ні на що не впливає, зараз, ми зберігаємо це і використовуємо в статистичних цілях. Як мінімум можна порівняти що користувач насправді використовує, та що обирав на початку.

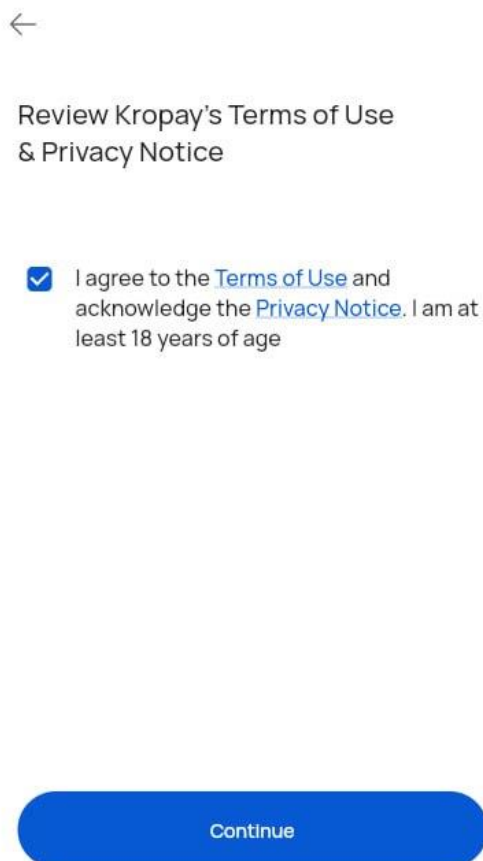


Рисунок 2.11 - Правила та умови

Даний крок є обов'язковим з декількох причин:

- **Юридичний захист:** Умови є юридично обов'язковою угодою між постачальником програми та її користувачами. Вони встановлюють правила та вказівки, яких повинні дотримуватися обидві сторони, захищаючи права програми та її користувачів.
- **Згода користувача:** Умови гарантують, що користувачі ознайомлені з умовами використання програми та погоджуються з ними. Приймаючи умови, користувачі дають свою інформовану згоду на політику та практику програми.
- **Обмеження відповідальності:** Умови та положення часто містять застереження та обмеження відповідальності, захищаючи постачальника програми від потенційних судових позовів або збитків, спричинених використанням програми. Ці положення можуть допомогти визначити ступінь відповідальності постачальника програми та мінімізувати юридичні ризики.

- Права інтелектуальної власності: програми часто включають інтелектуальну власність, наприклад вміст, захищений авторським правом, торговельні марки або патенти. Умови та положення допомагають визначити, як користувачі можуть або не можуть використовувати, відтворювати чи поширювати інтелектуальну власність програми, захищаючи права постачальника програми.

- Інструкції для користувачів: положення та умови можуть окреслювати прийнятну поведінку та правила використання для користувачів. Це допомагає підтримувати шанобливе та законне середовище в додатку, забезпечуючи дотримання чинних законів і запобігаючи зловживанням або неправильному використанню.

- Конфіденційність і захист даних: у зв'язку зі зростанням занепокоєння щодо конфіденційності та безпеки даних умови та положення часто містять положення щодо збору, зберігання та використання даних користувачів. Це гарантує, що користувачі отримують інформацію про те, як обробляються їхні дані, і забезпечує прозорість щодо практики конфіденційності».

Після реєстрації маємо користувача, який має доступ до головної сторінки застосунку, та до так званих захищених роутів, але що воно таке та як працює?



Рисунок 2.12 – Головна сторінка

Так звані *guards* є однією з фішок фреймворку Nestjs, виглядають як функції, які виконуються перед основною функцією та служать для захисту, повертають булеве значення, де 1 – доступ є, а 0 – немає [5].

В даному проєкті і три основних типів гуардів (*guards*):

- **Перевірка статусу користувача:** даний гуард відповідає за захист від частково зареєстрованих користувачів. Логіка доволі проста: користувач повинен пройти всі етапи реєстрації для доступу до основних методів застосунку.
- **Перевірка ролі користувача:** звичайна рольова система, поки що в застосунку є дві ролі *user*, *admin*. Звичайно що звичайний користувач не повинен мати доступу до адмін-панелі, або до функцій адміністратора.
- **Логічні перевірки:** даний вид гуардів виконує багато різноманітних захисних дій, наприклад при завантаженні картинок він перевіряє що це дійсно картинка, яка за розміром не перевищує N, тощо.

Також, треба згадати за рахунок чого застосунок розуміє який це користувач? Яка в нього роль? Який статус?

JWT – це відкритий стандарт, який визначає компактний і самодостатній метод передачі інформації між сторонами як об’єкт JSON. Він зазвичай використовується для автентифікації та авторизації у веб-додатках. Токен JWT складається з трьох частин: заголовка, корисного навантаження та підпису. Заголовок визначає алгоритм, який використовується для створення підпису, тоді як корисне навантаження містить претензії або твердження про користувача та інші відповідні дані. Підпис створюється шляхом поєднання закодованого заголовка та корисного навантаження з секретним ключем, що забезпечує засіб для перевірки автентичності та цілісності маркера. Токени JWT часто використовуються для безпечної передачі ідентифікаційних даних користувача та іншої відповідної інформації між клієнтом і сервером, дозволяючи автентифікацію без збереження стану та зменшуючи залежність від сховища сеансів на стороні сервера [6].

Не треба забувати що JWT токен це лиш інструмент, в нашому застосунку використовується лише 1, так званий, access token, але я також розробив і іншу версію застосунку, який використовує 2 токена, а саме: access, refresh. Головний плюс даного підходу в тому, що користувач отримує пару токенів, один з них використовується для доступу до захищених функцій, а інший – для перезапису пари ключів після закінчення строку дії. Зазвичай access токен не живе більш ніж 10 хвили, а refresh не має терміну придатності, але використати його можна лиш один раз.

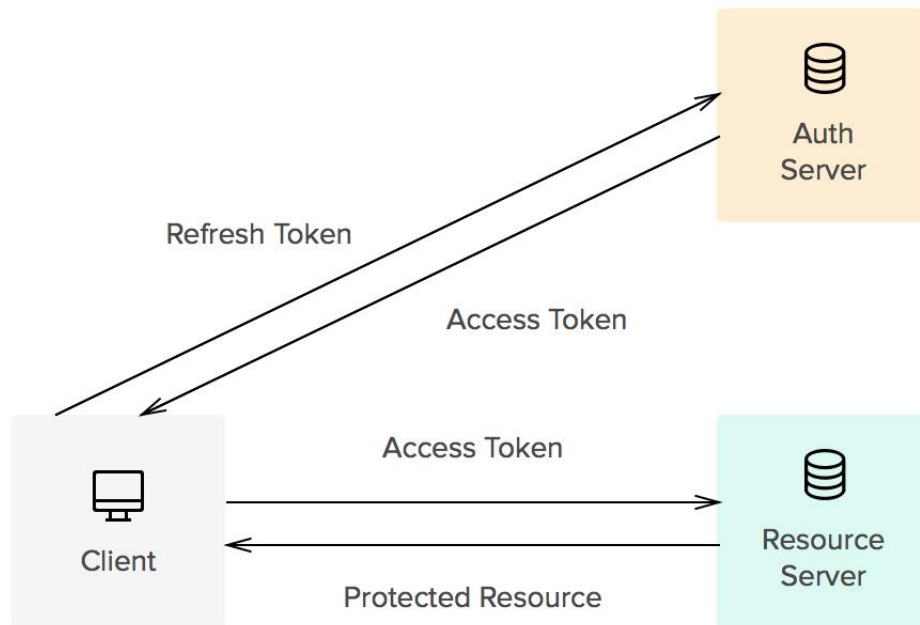


Рисунок 2.13 – Схема використання access, refresh токенів

2.4.2 Створення замовлення

Основна ідея електронної комерції – транзакції куплі/продажу, в нашому застосунку для даних маніпуляцій створено домен `order`, який відповідає за створення, обробку, маніпуляції з замовленням. Після натиску на `request payment`, користувач бачить наступне:

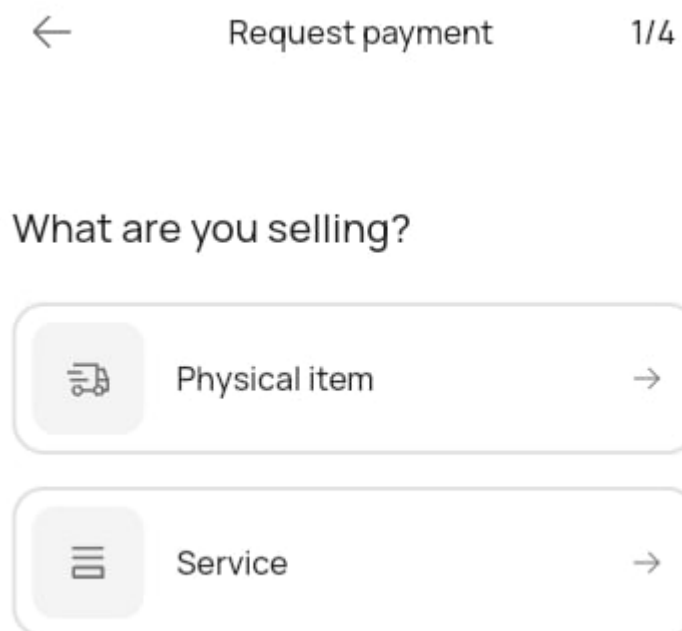


Рисунок 2.14 – Вибір типу замовлення

На даний момент у застосунку представлено лише 2 типи товару: фізичний предмет та сервіс. Фізичним предметом є будь-яка річ яку можна доставити доставкою, тоді як сервіс виступає у якості інформації, допомоги або роботи, яка виконується онлайн і не потребує доставки. Для кожного з цих видів відрізняються статуси та логіка. До прикладу фізичний предмет має такі статуси як: *'paymentRequested'*, *'paymentSecured'*, *'inProgress'*, *'dispatched'*, *'delivered'*, *'accepted'*, *'deliveryRejected'*, *'replaceItem'*, *'refund'*, *'cancelled'*, *'inDispute'*. Ці статуси служать для відслідковування положення, яке займає дане замовлення. Також, треба підмітити що встановлення статусів жорстко регламентоване та йде поступово, деякі статуси встановлює продавець, інші – покупець.

← Request payment 2/4

Tell us more about item you're selling

Title

Canon Camera

Description

Canon EOS R5 full-frame mirrorless camera

Add image

Click here to upload image
PNG or JPG

Continue

Рисунок 2.15 – Заповнення інформації замовлення

Після вибору типу замовлення, далі йде обов'язковий крок – заповнення деталей замовлення, а саме: заголовка, опису, фото (не обов'язкове). Данна інформація буде відображатись покупцю, коли він перейде на сторінку цього замовлення, також в застосунку передбачено функцію редагування.

← Request payment 2/4

Title

Clown Boots

Description

Fashion brand new clown boots, size between 37-50

Add image



Continue

Рисунок 2.16 – Повністю заповнене замовлення

Як ми бачимо, застосунок дозволяє додавати картинки, а з ними зв'язана цікава логіка. Треба відмітити що зберігати картинки можна або локально, або на хмарному сервісі по типу AWS, звичайно що локальне збереження використовується зачасту лише для тестових, або на маленьких проектах, в ньому є багато мінусів, основні з яких, те що це займає місце на машині на якій і працює сервер, а це потенційна загроза для атак, які мають за мету завантажити на 100% пам'ять і таким чином вивести з ладу сервер, але у даного підходу є і плюс, він дуже просто реалізується, для збереження картинки достатньо додати папку і завдяки вбудованій лібі Node.js – fs (file system), зберегти картинку за вказаним шляхом. Підхід з використанням стороннього API – надійніший, бо в такому разі ми маємо відокремлену логіку зі збереження картинок і сам сервер, але звичайно цей підхід не безкоштовний, тому треба платити за місце на дисках та за

потужності, що виділяються. Для хостингу сервера та картинок, я використовую портал DigitalOcean. Після тестів я і команда зрозуміла що картинки займають багато місця, тому було вирішено використати якийсь алгоритм для компресії картинок, мій вибір пав на JPEG алгоритм, з максимальною якістю 40% від початкової, готову реалізацію знайшов у бібліотеці sharp.

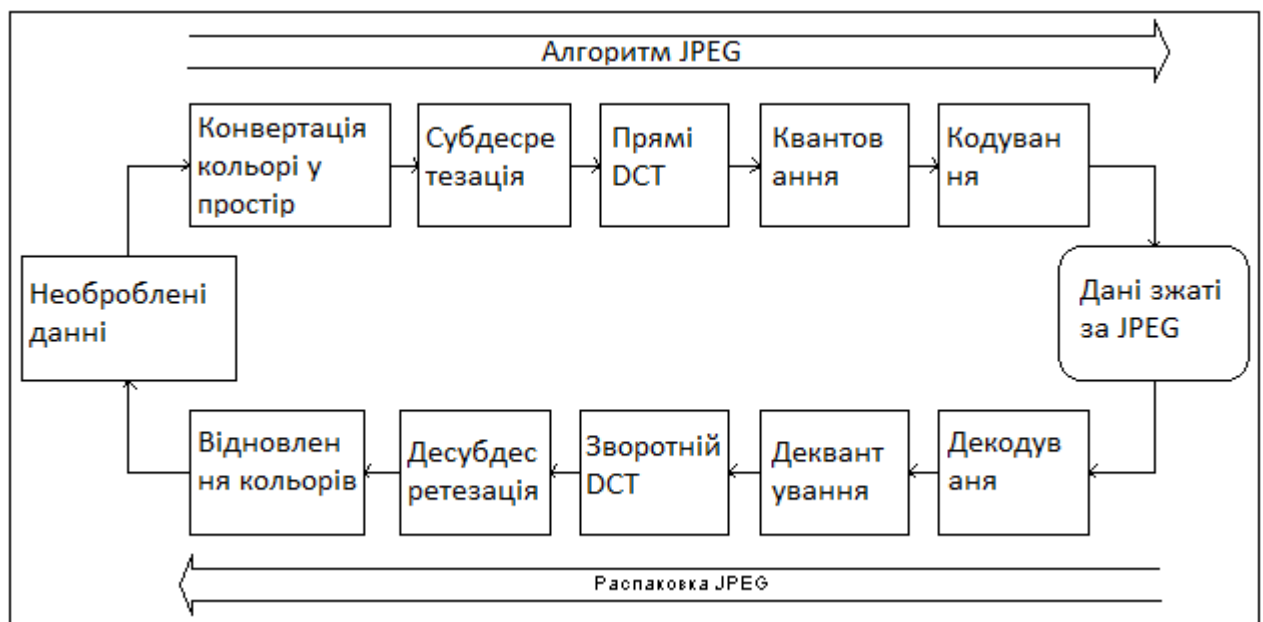
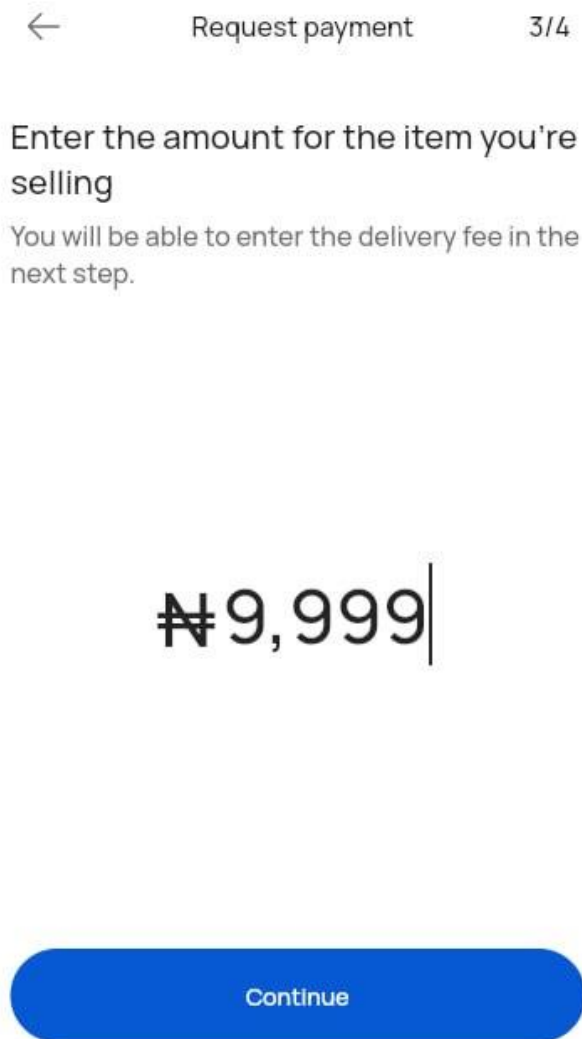


Рисунок 2.17 – Алгоритм JPEG

Протестувавши різні комбінації втрати якості, я, на втраті якості в 40% не помітив ніяких погіршень в картинці.

Далі, за логікою створення замовлення потрібно заповнити його ціну.



← Request payment 3/4

Enter the amount for the item you're selling

You will be able to enter the delivery fee in the next step.

¥9,999|

Continue

Рисунок 2.18 – Ввід ціни товару

Після заповнення ціни, опціонально можна ще додати ціну на доставку, яка буде складена з головною ціною при виставленні рахунку покупцеві.

Payment request sent



What's next?

↗ Buyer

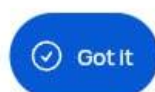


Рисунок 2.19 – Вікно відправки посилання на товар

Після заповнення ціни, для замовлення генерується зашифроване посилання на товар, по якому він може перейти та потрапити на сторінку товару, а потім і оплати.

Для шифрування посилання я використовую симетричний алгоритм з ключем. Алгоритм із симетричним ключем — це криптографічна техніка, яка використовується для шифрування та дешифрування даних, де для обох операцій використовується той самий секретний ключ. Він характеризується здатністю забезпечувати конфіденційність, цілісність і достовірність інформації, якою обмінюються сторони.

В алгоритмі симетричного ключа відправник використовує секретний ключ, щоб перетворити звичайний текст у незрозумілу форму, яка називається зашифрованим текстом. Цей зашифрований текст може бути розшифрований

назад у вихідний відкритий текст лише авторизованим одержувачем, який володіє тим самим секретним ключем.

Сильна сторона алгоритму з симетричним ключем полягає в секретності та складності самого ключа. Ключ має бути конфіденційним і безпечно передаватись між сторонами, що спілкуються. Чим більший розмір ключа та безпечніші методи керування ключами, тим стійкішим є алгоритм до криптографічних атак.

Завдяки використанню алгоритмів із симетричним ключем конфіденційну інформацію можна безпечно передавати та зберігати, захищаючи її від несанкціонованого доступу та забезпечуючи конфіденційність і цілісність даних.

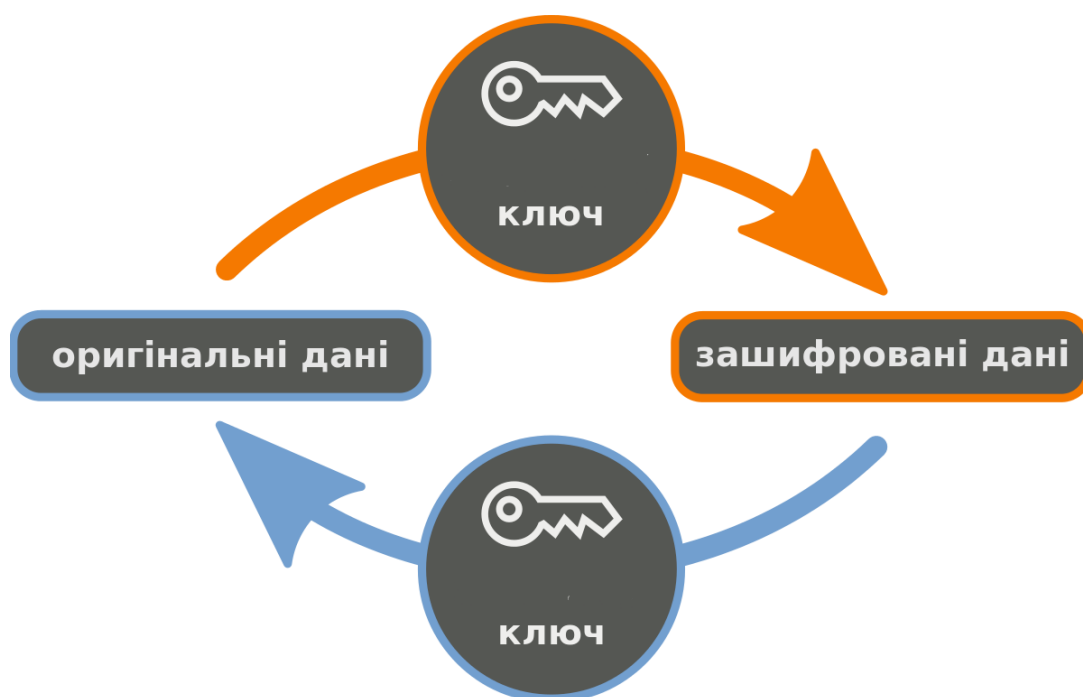


Рисунок 2.20 – Схема симетричного шифрування

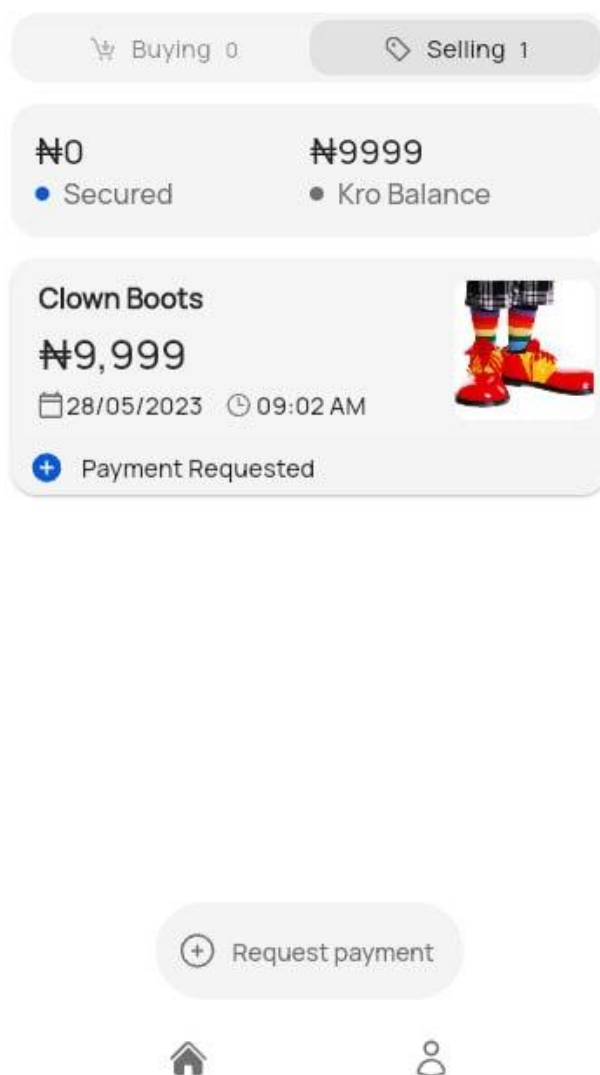


Рисунок 2.21 - Головна сторінка з замовленням

Після відправки посилання до покупця продавець бачить на головній сторінці замовлення яке він тільки що створив, його статус і іншу корисну інформацію, після підтвердження платежу покупцем – продавець отримає SMS + Email повідомлення.

ВИСНОВКИ

По завершенню даної дипломної роботи маємо готовий застосунок, який орієнтований на певну аудиторію. Даний застосунок дозволяє користувачам виконувати такі операції як:

- Реєстрація / авторизація
 - Створення продавцем замовлення, яке він продає
 - Створення покупцем замовлення для продавця з передоплатою
 - Користувач отримує вчасні повідомлення про зміну статусів замовлення, тощо.
-
- Оплату замовлення
 - Вивід грошей, якщо ви продавець
 - Додачу реквізитів на які буде скоєно payout

ПЕРЕЛІК ПОСИЛАНЬ

1. **Казакевич А.** E-commerce. Как завоевать клиента и не потерять деньги [Электронный ресурс] / А. Казакевич. – 2019. – №1. – С. 1–30. – Режим доступа до ресурсу: <https://kniga.biz.ua/pdf/25181-E-commerce.pdf>.
2. **Vaughn, V.** Implementing Domain-Driven Design [Электронный ресурс] / V. Vaughn. – Addison-Wesley Professional, 2013. – Р. 656. – Режим доступа: <https://dl.ebooksworld.ir/motoman/AW.Implementing.Domain-Driven.Design.www.EBooksWorld.ir.pdf>.
3. **Мартин, Р.** Чистая архитектура. Искусство разработки программного обеспечения [Электронный ресурс] / Р. Мартин. – СПб.: Питер, 2018. – 352 с. – Режим доступа: <https://monster-book.com/reader/22032>.
4. **Пауэрс, Ш.** Изучаем Node. Переходим на сторону сервера [Текст] / Ш. Пауэрс. – СПб.: Питер Пресс, 2017. – 304 с.
5. **Nestjs** [Электронный ресурс]. – Режим доступа: <https://docs.nestjs.com/>.
6. **Мартин, Р.** Чистый код. Создание, анализ и рефакторинг [Электронный ресурс] / Р. Мартин. – СПб.: Питер, 2013. – 464 с. Режим доступа: [https://sd.blackball.lv/library/chistyj_kod_-_sozdanie_analiz_i_refaktoring_\(2013\).pdf](https://sd.blackball.lv/library/chistyj_kod_-_sozdanie_analiz_i_refaktoring_(2013).pdf)

ДОДАТОК А

Вихідний код

Процес авторизації / реєстрації

```

async authentication(phoneNumber: string) {
  let savedUser = await this.userRepository.getByPhoneNumber(phoneNumber);

  if (!savedUser) {
    const defaultUserRole = await this.rolesRepository.getByRoleName(
      RoleNames.USER,
    );
    const newUser = new User();
    newUser.phoneNumber = phoneNumber;
    newUser.roleId = defaultUserRole.id;
    savedUser = await this.userRepository.save(newUser);
    await this.kroBalancePublicContract.createKroBalanceForUser(savedUser.id);
  }

  await this.verificationCodesPublicContract.sendLoginVerificationCode(
    savedUser.id,
    savedUser.phoneNumber,
  );
  return this.tokenClient.generateAuthenticationToken(savedUser.id);
}

```

Відправка СМС

```

private async sendSms(
  params: MessageListInstanceCreateOptions,
): Promise<void> {
  try {
    await this.twilioClient.messages.create({
      from: this.twilioPhoneNumber,
      body: params.body,
      to: params.to,
    });
  } catch (e) {}
}

async sendSmsWithCode(payload: SendSmsWithCodeDto): Promise<void> {
  const body = format(BodyTemplates.CODE, { code: payload.code });
  return this.sendSms({ to: payload.to, body });
}

```

Відправка електронних листів

```

private createParams(

```

```

    userEmail: string,
    templateId: string,
    dynamicTemplateData: Record<string, unknown>,
  ): MailDataRequired {
    return {
      to: userEmail,
      from: this.from,
      templateId,
      dynamicTemplateData,
    };
  }

  async sendVerificationEmail(
    userEmail: string,
    verificationCode: unknown,
  ): Promise<void> {
    const templateId = DynamicTemplates.EMAIL_VERIFICATION;
    const dynamicTemplateData = { code: verificationCode };

    await this.sendMail(
      this.createParams(userEmail, templateId, dynamicTemplateData),
    );
  }

```

Генерація та збереження OTP коду

```

private async createAndSaveVerificationCode(
  userId: number,
  type: VerificationCodesTypes,
): Promise<VerificationCode> {
  const verificationCode = new VerificationCode();

  verificationCode.code = this.generateVerificationCode();
  verificationCode.userId = userId;
  verificationCode.type = type;
  verificationCode.createdAt = new Date();

  return this.saveUniqueCodeWrapper(verificationCode);
}

private generateVerificationCode() {
  const rand = MIN_CODE + Math.random() * (MAX_CODE + 1 - MIN_CODE);
  return Math.floor(rand);
}

```

Оптимізація картинок перед збереженням

```

export const FileOptimization = async (value: Express.Multer.File) => {
  const { buffer } = value;
  const newBuffer = await sharp(buffer)
    .toFormat('jpeg')
    .jpeg({ quality: 40 })
    .toBuffer();
  value.buffer = newBuffer;
  value.mimetype = DefaultMimeType;
  return value;
};

```

Створення замовлення

```

async create(
  payload: CreateDto,
  sellerId: number,
  date: Date,
): Promise<Order> {
  const order = new Order();

  order.title = payload.title;
  order.price = payload.price;
  order.deliveryPrice = payload.deliveryPrice;
  order.images = payload.images;
  order.seller = sellerId;
  order.status = OrderStatuses.PAYMENT_REQUESTED;
  order.createdAt = date;

  if (payload.description) order.description = payload.description;
  if (payload.socialMediaLink)
    order.socialMediaLink = payload.socialMediaLink;

  return this.saveOrderWrapper(order);
}

private saveOrderWrapper(order: Order) {
  this.updateStatusHistory(order);
  return this.orderRepository.save(order);
}

private updateStatusHistory(order: Order) {
  if (!order.statusHistory) {
    order.statusHistory = [];
  }
  const lastRecord = order.statusHistory[order.statusHistory.length - 1];
  if (!lastRecord || lastRecord.status !== order.status) {
    order.statusHistory.push({ status: order.status, date: new Date() });
  }
}

```

Створення транзакції

```

async createTransaction(
  orderId: number,
  userId: number,
  date: Date,
): Promise<any> {
  const orderInfo = await this.orderPublicContract.getOrderForTransaction(
    orderId,
  );

  const feeConfiguration =
    await this.feeConfigurationRepository.getConfiguration();

  const transaction = new Transaction();
  const amount = this.amountCalculationLogic(orderInfo);

  transaction.externalRef = KroUtils.generateUniqueString();
  transaction.currency = orderInfo.currency;
  transaction.sellerId = orderInfo.sellerId;
  transaction.buyerId = userId;
  transaction.type = TransactionTypes.INCOME;
  transaction.orderId = orderId;
  transaction.createdAt = date;

  if (feeConfiguration.isBuyerHasFee) {
    transaction.fee = await this.calculateFee(amount);
  }

  this.calculateTransactionAmount(transaction, amount);

  const userResponse = await this.managementPublicContract.getUserById(
    userId,
  );

  const createPayment: any = {
    uniqId: transaction.externalRef,
    amount: transaction.amount,
    currency: transaction.currency,
    customer: {
      email: userResponse.email || 'fake@gmail.com',
      phone_number: userResponse.phoneNumber,
    },
  };

  await this.createTransactionWrapper(transaction);

  return this.bankClient.createPayment(createPayment);
}

```