Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему:

«Розробка веб-сервісу для захищених інтернет-платежів з інтеграцією платформи Stripe та налаштуванням механізму захисту 3D Secure»

Виконала студентка 4 курсу, гр.1КІТС-21б
спеціальності 125– Кібербезпека
Освітня програма – Кібербезпека
інформаційних технологій та систем
(шифр і назва напряму підготовки, спеціальності)

Порицька В.В
(прізвище та ініціали)

Керівник: д.ф. (PhD), доц., каф. МБІС
Салієва О.В
(прізвище та ініціали)
«____» _____ 2025 р.

Рецензент: к.т.н., доц., доцент каф. ОТ
Богомолов С.В
(прізвище та ініціали)
«____» _____ 2025 р.

Допущено до захисту
Голова секції УБ кафедри МБІС
д.т.н., проф. Яремчук Ю.Є.
“____” _____2025р.

Вінниця ВНТУ – 2025

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

Рівень вищої освіти І-й (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність 125 – Кібербезпека
Освітньо-професійна програма - Кібербезпека інформаційних технологій та систем

ЗАТВЕРДЖУЮ
Голова секції УБ, кафедра МБІС

д.т.н., проф. Яремчук Ю.Є.
" 20 " березня 2025 р.

З А В Д А Н Н Я
на бакалаврську кваліфікаційну роботу студенту
Порицькій Вероніці Володимирівній
(прізвище, ім`я, по-батькові)

1. Тема роботи Розробка веб-сервісу для захищених інтернет-платежів з інтеграцією платформи Stripe та налаштуванням механізму захисту 3D Secure
Керівник роботи д.ф., доцент каф. МБІС Салієва Ольга Володимирівна
(прізвище, ім`я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від " 20 " березня 2025року № 97
2. Термін подання студентом роботи " 06 " червня 2025року
3. Вихідні дані до роботи Електронні джерела, наукові статті та документи, повязані з темою. Існуюче програмне забезпечення, технічна документація до програмного забезпечення, що використовувалось при розробці.
4. Зміст текстової частини:
Для досягнення поставленої мети в роботі було здійснено аналіз сучасних платіжних сервісів, що використовуються для обробки онлайн-платежів, розглянуто принципи роботи технології 3D Secure та її еволюцію. Побудовано структуру веб-сервісу для обробки платежів із використанням платформи Stripe, спроектовано базу даних для зберігання інформації про транзакції та користувачів, а також розроблено інтерфейс користувача для взаємодії з платіжною системою. Та проведено тестування розробленого додатку.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов`язкових креслень)
Робота 3D Secure, процес вибору версії протоколу 3D Secure, взаємодія компонентів, ER-модель розробленої бази даних.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Розділ I	Салієва О.В. д.ф., доц. каф. МБІС		
Розділ II	Салієва О.В. д.ф., доц. каф. МБІС		
Розділ III	Салієва О.В. д.ф., доц. каф. МБІС		

7. Дата видачі завдання 20 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	23.03.2025	Виконано
2	Аналіз предметної області обраної теми	24.03.2025	13.04.2025	Виконано
3	Розробка алгоритму роботи	14.04.2025	27.04.2025	Виконано
4	Написання бакалаврської роботи на основі розробленої теми	28.04.2025	25.05.2025	Виконано
5	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025	Виконано
6	Виправлення, уточнення, корегування бакалаврської дипломної роботи	28.05.2025	02.06.2025	Виконано
7	Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025	Виконано

Студент _____ Порицька В.В
(підпис) (прізвище та ініціали)

Керівник роботи _____ Салієва О.В
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.056.523

Бакалаврська кваліфікаційна робота складається із 82 сторінок формату А4, на яких є 24 рисунка, 1 таблиця, список використаних джерел містить 26 найменувань.

Метою бакалаврської роботи є розробка веб-сервісу для безпечних інтернет-платежів, інтегруючи платіжну платформу Stripe та налаштовуючи механізм захисту 3D Secure.

Основна частина присвячена механізмам захисту та розроблених міжнародних стандартів у галузі онлайн-платежів. Розглянуто різні версії протоколу 3D Secure, зазначено їх переваги та недоліки. Розроблено архітектуру, базу даних й інтерфейс веб-сервісу, що було практично реалізовано. Також обґрунтовано вибір мови програмування та обрану систему управління базами даних.

Практична реалізаціє веб-сервісу охоплює інтеграцію Stripe Api та 3D Secure, для забезпечення додаткового рівня захисту транзакцій, а також реалізацію необхідного функціоналу для створення та обробки онлайн-платежів.

Ключові слова: онлайн-платежі, 3D Secure, Stripe, безпечні онлайн-платежі.

ABSTRACT

UDC 004.056.523

The bachelor's thesis consists of 82 pages of A4 format, which includes 24 figures, 1 table, and a list of references containing 26 titles.

The purpose of the bachelor's thesis is to develop a web service for secure online payments by integrating the Stripe payment platform and setting up the 3D Secure security mechanism.

The main part of the thesis is devoted to security mechanisms and developed international standards in the field of online payments. Different versions of the 3D Secure protocol are considered, their advantages and disadvantages are outlined. The architecture, database, and interface of the web service are developed and practically implemented. The choice of programming language and the selected database management system are also substantiated.

The practical implementation of the web service includes the integration of Stripe Api and 3D Secure to provide an additional level of transaction security, as well as the implementation of the necessary functionality for creating and processing online payments.

Keywords: online payments, 3D Secure, Stripe, secure online payments.

ЗМІСТ

ВСТУП

Актуальність. У сучасних умовах онлайн-платежі займають значну частку від усіх платіжних операцій, через швидкість, зручність та надійність, загалом тому, що значно спрощує повсякденне життя пересічного перехожого. Адже, щоб оплатити послугу чи надіслати кошти близькій людині, більше не потрібно приходити у відділення банку та очікувати своєї черги, натомість самостійно виконуємо потрібні операції будь-якої миті.

Проте із зростанням електронних платежів, збільшується і випадки шахрайства та кіберзагроз, що спрямовані або на процес передавання коштів, або ж на банківську інформацію. Ситуації коли стався витік інформації у банківському секторі ведуть до значних фінансових і репутаційних збитків.

Актуальність впровадження надійних механізмів захисту онлайн-платежів є одним із пріоритетним напрямків у сфері інформаційної безпеки у зв’язку із поширеністю. Тому було розроблено міжнародні стандарти та протоколи, які покликані вирішити цю проблему. Одним із таких протоколів є 3D Secure, який був розроблений для забезпечення додаткового рівня захисту.

Інтеграція платіжної платформи Stripe та запровадження механізму захисту 3D Secure допоможе підвищити рівень безпеки платежів, знизити ризик шахрайства та забезпечити довіру користувачів до онлайн-платежів.

Метою роботи є розробка веб-сервісу для безпечних інтернет-платежів з інтеграцією платіжної платформи Stripe та налаштуванням механізм захисту 3D Secure.

Задачі дослідження:

– проаналізувати сучасні платіжні сервіси;

– дослідити принципи роботи технології 3D Secure;

– розробити структуру веб-сервісу для обробки онлайн-платежів із використанням Stripe API та реалізувати механізм 3D Secure;

– провести тестування функціоналу платежів із 3D Secure у межах реалізованого веб-застосунку та описати сценарії взаємодії користувача з системою.

Об’єктом дослідження є процеси здійснення онлайн-платежів та їх захист у веб-сервісах.

Предметом дослідження є методи інтеграції платіжної платформи Stripe та механізму 3D Secure у веб-сервіс для забезпечення безпечних онлайн-платежів.

Новизна полягає у комплексному підході до інтеграції Stripe API та 3D Secure, а також у побудові сценаріїв взаємодії користувача, під час проходження автентифікації.

Практична цінність полягає у використанні розробленого рішення у електронній комерції.

Апробація. Тези доповідей із цієї тематики представлені на LII Всеукраїнській науково-технічній конференції факультету менеджменту та інформаційної безпеки [10]

1 АНАЛІЗ СУЧАСНИХ СЕРВІСІВ ОБРОБКИ ПЛАТЕЖІВ ТА ПРОТОКОЛІВ ЗАХИСТУ БАНКІВСЬКИХ РАХУНКІВ

У даному розділі буде розглянуто популярні сервіси обробки онлайн-платежів наведено їх недоліки та переваги. Також проаналізовано протокол 3D Secure та наведено його аналоги. Проаналізовано загрози для онлайн платежів, а також методи захисту від них.

1.1 Аналіз сучасних сервісів обробки платежів

Відповідно до досліджень частка онлайн-транзакцій зросла з 18,8% у 2021 році до 19,4% у 2023 році, а за прогнозом у 2027 році складе близько 22,6% [1]. Тому, щоб надати можливість користувачам здійснювати покупки онлайн ефективно і безпечно та задовільнити зростаючий попит було розроблено багато сервісів для здійснення онлайн-транзакцій та різноманітних протоколів для захисту даних банківських карток, транзакцій та паролів.

Одним з найвідоміших сервісів обробки онлайн-платежів є PayPal [2] – даний сервіс був відокремлений від популярної торгівельної онлайн-платформи eBay у 2015 році. Додаток є посередником між клієнтами та банком, робота сервісу полягає у тому, що користувач додає свою картку чи банківський рахунок, далі створюється рахунок PayPal, адже спершу всі кошти надходитимуть на даний рахунок, а потім користувач вирішує куди перевести кошти. Після чого користувач може здійснювати покупки чи продавати товари, причому усі транзакції обробляються PayPal, а не банком користувача. Проте сервіс стягує додаткову плату за користування, функцію миттєвого переказу та міжнародні перекази. Також плата може змінюватися в залежності від країни та тарифного плану, що для багатьох користувачів є великим недоліком сервісу.

Хоча PayPal офіційно представлений в Україні, але має певні обмеження, тому не можливо скористатися всіма перевагами та можливостями.

До недоліків сервісу можна ще віднести високу плату у порівнянні з іншими й відсутність даного сервісу у деяких країнах. Також користувачі зазначають непрозорість комісій, що для реального бізнесу є великою

проблемою. Також PayPal через популярність та велику кількість клієнтів не завжди може вчасно відреагувати на проблеми у проведені платежів кожного продавця.

До переваг відноситься можливість співпраці з терміналами та простата інтеграції з своїм сервісом.

Захист інформації відбувається завдяки шифруванню транзакцій та даних користувачів, запроваджено програми захисту покупців та продавців, що допомагає захистити фінансові інтереси усіх сторін, також сервіс відповідає вимогам стандарту PCI DSS.

Braintree – компанія заснована у 2007 році, згодом у 2013 сервіс придбав PayPal, проте на відміну від, PayPal, який націлений на широку аудиторію, Braintree спеціалізується на власників електронної комерції, тобто продавців товарів [3]. Тому додаток надає гнучку систему інтеграції з можливістю налаштування всього процесу оплати на одній сторінці без перенаправлень на стороні веб сторінки, що дозволяє розробити сучасний власний додаток електронної комерції з можливістю оплати різними сервісами.

Для захисту платіжної інформації сервіс використовує інтелектуальну автентифікацію та функцію 3DS. Надає інструменти для боротьби із шахрайством, а також токенізує мережу для безпечної передачі даних у мережі Інтернет та відповідає вимогам стандарту PCI DSS.

Ціноутворення відображено на офіційній сторінці, для банківських карток та цифрових гаманців комісія становить 2,5 відсотка від суми платежу та 0,49 долара за транзакцію [4]. Прозора плата за користування сервісом, захищеність сервісу та глибока кастомізація є перевагою, проте глибока кастомізація створює недолік у вигляді складної інтеграції сервісу до власних розробок. Також сервіс недоступний на території України та низки інших країн.

2Checkout (Verifone) – компанія була заснована у 2000 році та надає послуги для власників онлайн-магазинів та покупців [5]. Сервіс дозволяє обробити міжнародні перекази для великої кількості країн у всіх регіонах з можливістю оплати притаманному даному регіону способом, тому багато підприємців обирають платформу, як основу для платіжної системи бізнесу.

Також доступні різноманітні функції звітності, управління фінансами та контролю і моніторингу ризиків. Як і інші сервіси відповідає міжнародним вимогам для запобіганню витоку та шахрайству. Додаткові функції такі як звіт чи управління підписками залежать від тарифного плану, як і у випадку з PayPal, але має фіксовану плату коли продавець з покупцем знаходяться в одній країні – 3,5 відсотки від загальної суми та 0,35 долара за проведення платежу. Та у випадку коли покупець та продавець знаходяться у різних країнах комісія від загальної суми зростає на 5,5 відсотків, плата за проведення платежу залишається однаковою.

До переваг відноситься прозорість комісій, відповідність міжнародним стандартам захисту транзакцій, велика кількість способів оплати та підтримка багатьох країн. Проте до недоліків за відгуками користувачів відноситься погана якість підтримки та утримання коштів.

Adyen– нідерландська компанія була заснована у 2006 році, пропонує ефективні рішення для обробки платежів як у онлайн форматі, так і в офлайн та дозволяє об’єднати ці два потоки у одне ціле, тому сервіс зарекомендував себе з позитивного боку для великих компаній та невеликих стартапів [6]. Адже платформа надає можливість глибоких налаштувань в залежності від клієнтських потреб. Надає індивідуальні тарифні плани, базуючись на потребах кожного підприємства, таким чином забезпечуючи конкурентоспроможність свого сервісу щодо тарифних планів. Згідно вказаної інформації на офіційному сайті платформа відповідає 1 PCI DSS, використовує токенізацію, новітні протоколи шифрування та технологію 3D Secure для підвищення рівня безпеки своїх послуг. Проте сервіс доступний лише у Західній Європі, Північній Америці, Австралії, Японії та ще у декількох розвинутих країнах. Для решти світу платформа недоступна в тому числі для України.

Особливістю даного сервісу є процес реєстрації для індивідуального продавця, щоб отримати обліковий запис спочатку необхідно подати заяву у якій буде зазначено бізнес-модель та країни у яких клієнт працює та приймає платежі. Якщо попередні дані підходять відповідно до політки Adyen далі створюються

інтеграція зі власним сервісом у тестовому режимі, а потім подається заява на активацію облікового запису після чого можна приймати реальні платежі [7].

Перевагами є індивідуальні тарифні плани, широкі можливості налаштувань безпеки обробки транзакцій. Недоліками є малий регіон охоплення.

Stripe – це платформа обробки платежів, яка дозволяє приймати платежі від клієнтів. Хоча ідея створення послуги обробки онлайн-платежів на момент заснування була вже не нова, але компанії вдалося досягти значних успіхів завдяки тому, що було розроблено платформу, яку було легко інтегрувати у свою розробку, на відмінно від своїх конкурентів. Також орієнтація на малі та середні бізнеси напочатку свого шляху зробила сервіс вибором номер один [8].

Вагомою перевагою даного сервісу є можливість впровадити платіжний шлюз не маючи спеціалізованих знань з програмування чи будь-яких інших, для таких користувачів розроблено графічний інтуїтивний інтерфейс за допомогою, якого можливо зручно налаштовувати необхідні параметри. Проте для більш просунутих користувачів, кому потрібно більше унікальних налаштувань існує Stripe API та документація до даного сервісу. Розширена звітність і аналітика, безперешкодна інтеграція із платформами електронної комерції та міжнародна підтримка ще одні з сильних сторін платформи. До недоліків відноситься сувора політика компаній з якими він співпрацює.

Тарифи залежать від способу оплати, до прикладу для банківських карт та цифрових гаманців, як Google Pay або Apple Pay комісія становить 2,9 відсотка від загальної суми та 0,3 долара за транзакцію, тобто тарифи на рівні конкурентів.

Як і інші подібні сервіси має допоміжні функції такі, як запобігання шахрайству з вбудованою системою безпеки – Stripe Radar, автоматизоване управління підписками. Надає можливість користуватися терміналами, що дозволить поєднувати онлайн та офлайн платежі у одному сервісі. Також має можливість ідентифікувати особистість користувачів через документи або біометрію.

Для забезпечення безпеки Stripe дотримується стандартів PCI DSS, також має сертифікат EMVCo для карткових терміналів, підтримує багатофакторну

автентифікацію, алгоритм одноразового паролю на основі часу та 3D Secure першої та другої версії.

Попри відсутність офіційної підтримки в Україні, компанії можуть легально використовувати Stripe через програму Stripe Atlas або реєстрацію у країнах-партнерах.

Для візуалізації порівняння різних сервісів обробки платежів створимо таблицю 1.1, в якій буде продемонстровано переваги та недоліки розглянутих сервісів.

Таблиця 1.1 – Порівняння сервісів обробки платежів

	PayPal	Braintree	2Checkout	Adyen	Stripe
Робота в Україні	+-	-	+	-	+-
3D Secure	+	+	+	+	+
PCI DSS	+	+	+	+	+
Простота інтеграції	+	-	+	+	+
Прозорість та доступність комісій	-	+	-	+	+
Робота у багатьох країнах	+	-	+	-	+
Режим тестування	+	+	+	+	+
Якість підтримки клієнтів	-	-	-	+	+

Проаналізувавши всі переваги та недоліки сервісів для роботи було вирішено обрати сервіс обробки платежів Stripe, оскільки він один з найпоширеніших платіжних сервісів у світі та наразі активно ведуться переговори, щодо офіційної підтримки в Україні, то вважаю дослідження даного сервісу актуальним. Також важливо зазначити, що у порівняні з іншими сервісами Stripe забезпечує розширений захист від шахрайства та загроз.

Отже, у даному підрозділі було здійснено порівняння різних сервісів обробки платежів. Порівняння було проведено серед ключових особливостей кожної платформи. У результаті обрано найбільш надійний за ключовими

параметрами сервіс для майбутньої успішної роботи та функціонування підприємства.

1.2 Особливості технології 3D Secure

У сучасному світі все більше платежів здійснюється онлайн, адже для користувачів це дозволяє швидко купувати необхідні товари в мережі, зручно та безпечно вводити власну платіжну інформацію і не замислюватись над тим, як платіж відбувається на стороні сервера. Для продавців онлайн платежі забезпечують легку звітність та аналітику, їм більше не потрібно тримати цілий штаб бухгалтерів, щоб обраховувати витрати та прибутки, тепер ця вся інформація створюється автоматично і в декілька кліків.

Проте збільшення кількості активних платоспроможних користувачів, активно викликає інтерес зловмисників та шахраїв. Для забезпечення безпеки платіжних даних користувачів розроблялися певні протоколи, які виконували дане завдання. Одним з перших протоколів захисту інформації був SSL, що базується на шифруванні даних між браузером та сервером.

Також для підтвердження фізичного доступу до картки, почали впроваджувати CVV коди на звороті кожної банківської карти, для зменшення шахрайства у мережі. Використовували різні варіації додаткової авторизації такі, як введення унікального пароля, одноразового пароля тощо. Все це стало основою для розробки удосконалених протоколів, як 3D Secure.

3D Secure – протокол, що був розроблений, щоб забезпечити додатковий рівень захисту для онлайн транзакцій з дебетових чи кредитних карт. Створений Arcot Systems (зараз CA Technologies) на замовлення організації EMVCo та вперше був застосований Visa для покращення безпеки онлайн платежів [9].

Назва базується на тому, що для роботи протоколу необхідно 3 домени для успішного проведення транзакції:

– банк і продавець, які будуть отримувати кошти за надані послуги;

– видавець, банк який видав кредитну або дебетову карту;

– домен сумісності, інфраструктура, яка надається карті для підтримки протоколу 3D Secure.

Тобто за допомогою XML та SSL, відбувається обмін даними між банком покупця та банком продавця. XML використовується, як стандартний формат обміну повідомленнями між всіма учасниками, а SSL/TSL шифрує інформацію для захисту даних під час автентифікації. Використовуючи даний протокол забезпечується захист від модифікування та перехоплення у той час поки банки обмінюються даними.

Таким чином відповідальність за повернення коштів чи підтверджену шахрайську операцію відповідальність несе банк видавця. Для власників карт Visa існує захист від шахрайства із поверненням платежу.

Для користувача процес роботи протоколу 3D Secure виглядає наступним чином:

– вводяться банківські дані для покупки товару чи послуги;

– після підтвердження відкривається сторінка банку, яким користується користувач;

– необхідно увійти у банківську систему зручним способом та підтвердити покупку;

– відкривається сторінка підтвердження чи не підтвердження транзакції.

Таким чином забезпечується додаткова перевірка, яка дозволяє впевнитися чи дійсно користувач має доступ до даної банківської карти. На практиці це означатиме, що навіть, якщо шахраю вийде отримати фізичний доступ до карти, він не зможе пройти дану перевірку, адже не знає пароль для проходження аутентифікації.

Проте для сервера все складніше [10]:

– покупець формує список товарів, що бажає придбати та натискає кнопку “оформити”;

– веб-сторінка ініціює запит з використанням протоколу 3-D Secure та надсилає платіжні дані до домену емітента, щоб автентифікувати покупця;

– банк-емітент перевіряє отриману інформацію на достовірність та у результаті надсилає відповідь щодо дозволу проведення платежу;

– якщо банк-емітент дозволив проведення платежу, клієнта буде переведено на іншу сторінку, де необхідно ввести пароль;

– отриманий пароль знову перевіряється банком-емітентом;

– якщо пароль правильний, то продавцю надсилається код авторизації, що підтвердить особу користувача;

– банк-еквайер завершує фінансову операцію, після чого покупка завершена.

Рисунок 1.1 – Робота 3D Secure

Значними перевагами даного протоколу є те, що він захищає як продавців так і клієнтів, а також дозволяє зняти з себе ризики незапланованих витрат чи втрати прибутку з продавців, проте велика кількість пересилань між адресами та тривалість процесу змушує деяких користувачів запідозрити шахрайські дії, або відмовитись від придбання обраного продукту. Тому було розроблено протокол 3D Secure 2.

3D Secure 2 покликаний покращити користувацький досвід та залишитись надійним способом перевірки користувачів. Забезпечується це тим, що замість статичного паролю у попередній версії використовується біометрична

аутентифікація на основі токенів, тобто відбиток пальця, розпізнавання голосу чи обличчя .

Дана версія підтримує автентифікацію на основі ризиків – це просто процес визначення ризику, пов’язаного з конкретною транзакцією, і, виходячи з рівня ризику, визначення того, чи потрібно користувачеві пропонувати додаткові кроки автентифікації. Складові ризику включають у себе вартість транзакції, історію транзакцій та поведінки клієнта [11]. Якщо сукупний ризик буде низьким через те, що користувач не новий, у нього багато успішних транзакцій та непідозріла поведінка, то приймається рішення не проводити додаткову перевірку. Якщо ж якийсь із даних показників є підозрілим, або новим – проводиться додаткова перевірка, адже ризик буде високим або середнім.

Таким чином забезпечується покращений користувацький досвід та більший прибуток для продавців. Також перевагою даної версії є можливість його використання не тільки для оплати у браузерних версіях веб-сайтів, а також у додатках на смартфоні та додатково підтримуються цифрові гаманці. Тому продавців, які хочуть використовувати додатковий рівень захисту не обмежується малим вибором платформи на якій він зможе розробити власний магазин, покупці також будуть користуватися додатками на тій платформі на якій їм комфортно. Та здійснювати транзакції не виходячи із додатку, що збільшить шанс успішного завершення оформлення замовлення.

Як результат роботи протоколу задоволеними залишаються усі, адже користувачам зручніше здійснювати покупки, а для продавців більший прибуток та менше ризиків.

Оскільки банківські установи підтримують різні версії протоколу, то постає питання вибору з якою версією необхідно працювати. Для вирішення цієї проблеми було спроектовано алгоритм, що представлений на рисунку 1.2 [12].

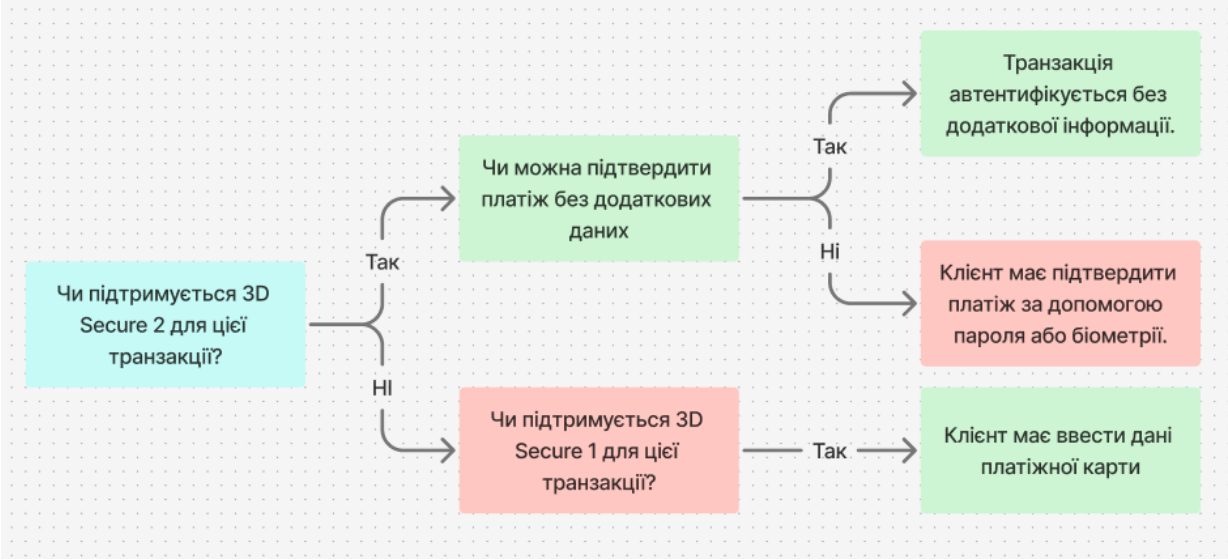


Рисунок 1.2 – Процес вибору версії протоколу 3D Secure

Алгоритм працює запитуючи спершу чи підтримується новіша версія, тобто друга версія.

Якщо друга версія підтримується далі йде перевірка чи можна підтвердити платіж без додаткових даних від користувача, тобто відбувається автентифікація на основі ризиків у результаті якої отримуємо позитивну чи негативну відповідь та виводимо користувачу відповідну відповідь. Також автентифікацію на основі ризиків часто називають Dynamic 3D Secure.

Якщо ж не підтримується друга версія, відсилається запит на підтримку першої версії та відбувається традиційна аутентифікація згідно якої користувач для успішного проходження транзакції повинен пройти додаткову перевірку.

Проте, як зазначалося раніше, додаткові перевірки спонукають багатьох користувачів покинути сторінку продавця, через це продавці змушені приймати певний компроміс між захищеністю від шахрайства та власним прибутком.

Останнє оновлення відбулось 29 вересня 2022 року EMVCo, глобальна технічна організація, оприлюднила оновлення специфікацій EMV® 3-D Secure (3DS), спрямоване на розширення можливостей емітентів і продавців для боротьби зі зростаючими ризиками шахрайства з використанням карток, які відсутні. EMV 3DS 2.3.1 базується на попередніх версіях специфікацій, представляючи нові елементи даних і потоки, які оптимізують автентифікацію

споживачів і посилюють заходи запобігання шахрайству за відсутності карток [13].

EMVCo – це консорціум із стандартизації у сфері електронних платежів, учасниками якого є шість найбільших платіжних систем: Visa, MasterCard, JCB, American Express, China UnionPay і Discover [14]. Організація відповідає за впровадження та оновлення протоколу 3D Secure.

Друга версія протоколу відповідає стандартам PSD2, тобто вимогам європейської директиви про платіжні послуги, що зробило використання даного протоколу обов'язковим на території Європейського Союзу, а також низки інших країн.

Протокол відповідає вимогам SCA (Strong Customer Authentication), тобто надійної автентифікації, що також є європейською законодавчою вимогою, впроваджена для боротьби з кіберзлочинністю. На сьогодні протокол є найпоширенішим способом перевірки особи власників карток, які здійснюють покупки в мережі відповідно до стандартів SCA [15].

Але, як і будь-яка технологія протокол має свої недоліки. Насамперед це складність і дороговартість впровадження, адже робота потребує значних зусиль та технічної інфраструктури відповідних знань. Тому впровадження та підтримка для малого і середнього бізнесу може виявитись важким завданням, що робить непривабливим протокол для таких компаній. Також проблема полягає у тому, що різні банки та платіжні шлюзи реалізовують з відмінностями підтримку протоколу, тому у процесі можуть виникнути певні помилки, що означатиме для користувачів неможливість здійснити покупку, відповідно для бізнесу втрату коштів. Робота вимагає обробки великої кількості конфіденційних даних, тому стандартних заходів безпеки інформації буде недостатньою і потребує залучення інших заходів для підвищення захисту від модифікації чи несанкціонованому перегляду. Протокол не здатний захистити у випадку атак фішингу чи соціальної інженерії.

Отже, у даному підрозділі розглядалась технологія захисту від шахрайства 3D Secure від початку створення та її призначення до розвитку у сучасні версії, а

також її переваги та недоліки, різниця між різними версіями. Використання у сучасному світі та важливість.

1.3 Сучасні виклики та загрози онлайн платежам і стандарти їх захисту

Кожного дня кількість загроз та ризиків інформації у мережі Інтернет зростає, тому необхідно постійно працювати над покращенням існуючих систем безпеки, щоб захистити дані від нових і нових ризиків. Особливо це стосується критичних систем таких як лікарські установи, системи опалення та електрозабезпечення, банківські установити та багато інших.

Оскільки все більше послуг переходять у онлайн, то більше і конфіденційних даних необхідно передавати через мережу. Особливо це стосується платежів у мережі, адже майже всі послуги необхідно оплачувати.

Перш ніж обговорювати загрози та вразливості, необхідно зрозуміти, як відбуваються платежі у мережі.

Цифрові платежі працюють наступним чином [16]:

– спершу користувач має розпочати платіж обравши оплату необхідної послуги зручним способом;

– платіжний шлюз шифрує дані транзакції та передає платіжному процесору;

– платіжний процесор отримує зашифровану інформацію та передає банкові користувача запит на ініціювання платежу;

– банк користувача отримує дані про ініціювання платежу та перевіряє дані користувача, суму коштів на рахунку та суму платежу, якщо сума платежу не перевищує суму на рахунку, платіж підтверджується та надсилається відповідь платіжному процесору;

– платіжний процесор після отримання позитивної відповіді від банку клієнта надсилає та перевіряє деталі транзакції до банку отримувача;

– якщо перевірка платіжним процесором пройшла успішно, то кошти перераховуються з одного банку до іншого.

– банк одержувача надсилає інформацію про отримання коштів до платіжного процесора, а він у свою чергу, надсилає таке ж повідомлення про успішне надходження коштів, банку клієнта та транзакція завершується.

– після чого, платіжний процесор надсилає дані про завершення платежу на платіжний шлюз, а далі дані надходять користувачу у вигляді сторінки підтвердження.

Виходячи з процесу обробки цифрових платежів, банк одержувача та клієнта не спілкується між собою напряму, а між ними присутній платіжний процесор, який відповідає за обробку платіжних транзакцій, до прикладу Visa та Mastercard. Та платіжний шлюз, що забезпечує шифрування чутливої інформації покупця та деталі транзакції. Прикладами платіжного шлюзу можуть бути розглянуті сервіси у першому підрозділі даного розділу, але всі вони одночасно виконують роль платіжного процесору. Рішення об’єднувати платіжний шлюз та процесор набуває все більше популярності, адже це економить кошти, полегшує інтеграцію до власного сервісу, через те, що не потрібно витрачати сили на налаштування роботи одним з одним, відповідно ризик помилки при передаванні даних знизиться. Адже сервіси з самого початку розроблялися, як цілісний продукт.

У реальності цифрові платежі відбуваються від кількох секунд до кількох хвилин в залежності від швидкості мережі Інтернет, але за вказаний проміжок часу сервіси повинні здійснювати багаторазовий обмін даними. Відповідно це створює ризики перехоплення, модифікації чи повного знищення інформації, тому міжнародні банківські організації розробили ряд стандартів, протоколів та правил для забезпечення максимальної безпеки даних під час процесу транзакції.

Один з найпоширеніших стандартів захисту даних карток під час їх функціонування – PCI DSS (Payment Card Industry Data Security Standard). Першу версію було випущено у 2006 році на основі систем безпеки Visa та Mastercard. Згодом стандарт значно змінювався у відповідь на нові ризики та вимоги міжнародної спільноти.

Остання четверта версія PCI DSS вийшла у березні 2022 року, до даної версії було додано ряд нових вимог, щоб відповідати на нові виклики, особливо

це стосується тих загроз, які спрямовані на постачальників банківських послуг. Також зазначається, що всі оцінювання, які розпочалися пізніше 1 квітня 2024 року повинні базуватися на четвертій версії стандарту PCI DSS, повний перехід з версії 3.2.1 планується завершити у 2025 році [17].

Великі терміни впровадження обумовленні суттєвими змінами у багатьох сферах захисту даних, тому для організацій потрібно багато часу, щоб впровадити зміни та протестувати їх. Зміни насамперед стосуються запровадженню нових алгоритмів на сторінці платежу, алгоритм повинен виявляти неочікувані зміни роботи основного функціоналу сторінки та негайно повідомляти про це адміністраторам. Це дозволить швидко виявити спробу впровадження шкідливого коду та заборонити користувачам перейти за підробленим посиланням, таким чином забезпечить безпеку платіжних даних користувача та збереже репутація бізнесу. Надалі багатофакторна автентифікація буде потрібна всім користувачам, а не лише адміністраторам. Крім того, мінімальну довжину пароля збільшено з 8 до 12 символів, що допоможе посилити захист та завадити несанкціонованому доступу.

Стандарт запроваджує щорічну перевірку на предмет відповідності вимогам, а також навчання співробітників, яке базується на поточних загрозах.

Особливістю PCI DSS є його глобальний характер, вимогам якого мають відповідати усі установи, що здійснюють платіжні операції. З одного боку міжнародний стандарт надає перевагу, через відсутність проблем з сумісністю чи важкістю інтеграції, але з іншого боку це вимагає прискіпливої перевірки та обговорення кожної деталі, а також неможливість запровадження індивідуальних правил, які могли б, бути корисним для певного регіону чи країни.

Стандарт, що працює на території європейських країн – GDPR (General Data Protection Regulation). Хоч це не суто платіжний стандарт, проте стосується захисту персональних даних користувачів.

Правила базуються на принципах, порушення яких призведе до штрафу в розмірі до 4% від загальнорічного світового прибутку [18]:

– законність, чесність та прозорість. Визначає, що дані повинні оброблятися за умови згоди та зрозумілого оповіщення користувача про те, яка інформація збирається та з якою метою;

– обмеження призначення. Інформація повинна збиратися лише для заздалегідь визначених та законних цілей, причому ціль має бути чітко сформульована, щоб перевірити законність обробки даних. Також принцип вимагає відповідності чинним законодавчим вимогам усіх рівнів;

– мінімізація даних. Збір конфіденційних даних не повинен бути надмірним, а лише відповідати поставленій меті. На цьому етапі користувачі мають задатися питанням чи потрібні ті дані, що вони передають для цілі сервісу обробки. Якщо необхідно надати дані, які прямо не стосуються потрібної послуги, то не варто співпрацювати із платформою;

– точність. Потрібно докладати зусиль, щоб дані регулярно оновлювались та в будь-який момент відповідати реальності, для того, щоб не зіштовхуватись із правовими наслідками та працювати із правильними даними. Якщо виникають сумніви у актуальності чи правильності, необхідно вжити всіх заходів для їх актуалізації. Якщо оновити інформацію неможливо – видалити її;

– обмеження зберігання. Персональні дані повинні зберігатися не довше ніж це необхідно для досягнення мети. Термін зберігання має відповідати допустимому мінімуму. Це обмеження було встановлено через недостатню кількість серверів та коштів на утримання. Тому, щоб раціонально використовувати наявні ресурси, а не просто збільшувати обсяги серверів, потрібно видаляти ті дані, які більше непотрібні для обробки.

– цілісність та конфіденційність. Обробляючи персональні дані потрібно дотримуватись такого способу, який забезпечить їх безпеку. Це стосується несанкціонованого доступу та обробки, втрати з будь-яких причин та пошкодження.

Розглянуті принципи допомагають забезпечувати чіткі й структуровані правила для забезпечення максимального рівня захисту прав користувачів. Встановлюють відповідальність за несанкціоновану обробку персональних

даних, що зобов’язує організації відповідально ставитись до конфіденційної інформації та запроваджувати контроль і навчання для співробітників.

Вагомими пунктами у стандарті представлено забезпечення прав суб’єктів, тобто користувачів, а також передання персональних даних третім країнам, що не знаходяться у ЄС. Дані пункти розроблені для того, щоб зберегти та захистити персональні дані громадян, а також забезпечити користувачам обирати чи надавати інформацію для обробки, мати можливість дізнатися про мету та спосіб обробки даних. Особливо передача конфіденційної інформації третім країнам, вимагає спеціальних заходів для безпеки, оскільки у третіх країнах можуть не бути прийняті високі стандарти захисту і прав для суб’єктів. Згідно даного стандарту, щоб передавати за межі ЄС персональні дані, необхідно довести, що у країні прийняті стандарти, які забезпечують відповідний рівень або організація, що буде обробляти інформацію має надійні стандарти обробки персональних даних.

Стандарт запроваджує єдині високі вимоги для захисту інформації користувачів та вводить механізми регулювання, для перевірки відповідності вимогам.

У сфері фінансових стандартів широко відомий стандарт PSD2. Відмінність між стандартом PCI DSS полягає у тому, що PCI DSS спрямований на захист дебетових та кредитних карток та операцій між ними, а PSD2 – європейський стандарт для банківських установ і тих компаній, що надають послуги відкритого банкінгу.

Перший стандарт був випущений 2007 року, тож достатньо швидко застарів, через стрімкий розвиток інформаційних технологій і відповідно банківських послуг. Зокрема деякі нові учасники були поза PSD 1, а також вимоги стандарту не могли повною мірою забезпечити безпеку банківському сектору. Тому було створено другу версію. Мета другого стандарту PSD полягає у покращеному механізму захисту платежів та даних клієнтів, сприяння інновацій у банківській сфері послуг та забезпечені відповідних правових умов для передових організацій, які працюють у цій сфері [19].

З найбільших нововведень – дозвіл третім сторонам отримувати доступ до банківських рахунків користувачів через відповідні банківські інтерфейси для надання додаткових фінансових послуг користувачам. Наприклад сюди відносяться популярні сервіси, які збирають інформацію про банківський рахунок з різних банків та збирають ці звіти у одному місці, таким чином користувачу не потрібно моніторити стан рахунку одразу у декількох банках, а все зручно буде доступно у одному місці. Також сервіси ініціюють платежі від імені користувача, таким чином підтримуються інновації у сфері, що у майбутньому спричинить ще більше покращень.

Зрозуміло, що для безпечної роботи таких сервісів, необхідно посилити захист. Для цієї мети було введено надійну аутентифікацію користувачів та алгоритми захищеного з'єднання для інтерфейсів.

Окрім вищерозглянутих стандартів існують також інші – Swift, ISO 20022, AML та KYC та інші. Всі забезпечують стандартизацію та правову основу у банківській сфері для конкретних цілей, як обмін повідомленнями між банківськими установами, створення та робота цифрових підписів чи робота банківської міжнародної мережі. У всіх цих питаннях важливу частину завжди буде займати захист інформації, необхідний у процесі роботи.

Проаналізувавши стандарти у фінансовій сфері, можна зробити висновок, що вони зосереджені на забезпеченні захисту персональних даних на всіх етапах їх передачі та обробки у мережі Інтернет. Стандарти, такі як PSD2, PCI DSS та GDPR, створені прямо чи опосередковано для забезпечення безпеки, конфіденційності та цілісності фінансових даних у цифровому середовищі

1.4 Висновки до розділу 1 та постановка задач

У цьому розділі розглядаються ключові аспекти обробки онлайн-платежів, зокрема через популярні платіжні сервіси Stripe, PayPal та інші, а також важливу роль, яку відіграє 3D Secure у забезпеченні безпеки платіжних транзакцій. Дослідження платформ обробки цифрових платежів демонструють, що вони відіграють важливу роль у повсякденному житті користувачів. Їх переваги у

сфері фінансових послуг – простота інтеграції, підтримка кількох методів оплати та надійний захист даних клієнтів.

Актуальною проблемою онлайн-платежів є забезпечення безпеки при їх обробці, для цього було запропоновано міжнародною спільнотою низку стандартів та протоколів. Однак незважаючи на впроваджені механізми захисту існують виклики пов’язані із шахрайством та крадіжкою даних, відсутність повної відповідності стандартам PCI DSS та GDPR, що створює ризики для безпеки даних користувачів.

Ґрунтуючись на результати дослідження та актуальності теми було поставлено такі задачі:

- розробити архітектуру, базу даних та інтерфейс веб-сервісу;
- обґрунтувати вибір мови програмування для втілення сервісу;
- реалізувати продукт;
- протестувати розроблений веб-сервіс.

У наступних розділах планується досягти поставлених задач та мети роботи.

2 РОЗРОБКА АЛГОРИТМУ ТА СТРУКТУРИ ЗАХИЩЕНОГО ВЕБ-СЕРВІСУ

Перед безпосередньою розробкою додатку важливо спершу спроектувати основні компоненти, продумати їх взаємодію та роль. Також варто звернути увагу на інтерфейс користувача, адже потенційні покупці будуть співпрацювати з ним. Необхідно продумати всі кроки користувача та зробити їх максимально легкими та зрозумілими.

2.1 Розробка архітектури веб-сервісу

Будуючи систему електронної комерції важливо забезпечити надійність та захищеність тому, що для користувачів дані критерії є досить важливими під час роботи з будь-якими додатками. Основна мета проектування архітектури – розробити таку модель, яка б відповідала цим вимогам та забезпечувала достатню швидкодію при цьому.

Умовно весь сервіс можна поділити на декілька компонентів, які разом забезпечать повноцінний функціонуючий додаток. Перший компонент – фронтенд частина, яка відповідає за взаємодію з користувачем. Для успішного додатку електронної комерції користувацька частини повинна бути не просто привабливою, а й інтуїтивно зрозумілою та не перевантажена зайвими елементами дизайну чи надмірною кількістю функцій. Важливо створити безпечні форми для збору даних користувачів, візуальні пояснення чи підказки, щоб користувачі розуміли, що від них вимагається.

Другим компонентом є серверна частина або бекенд. Ця частина повинна об’єднатися із фронтендом та безпечно передавати інформацію користувача для обробки стороннім сервісам. Також має бути реалізація інтеграції із базою даних. Оскільки робота стосується захищеного сервісу обробки онлайн платежів, додатково необхідна взаємодія з сервісом, що виконує дане завдання.

Третій компонент – обробка результатів 3D Secure, так як у результаті роботи даного протоколу, можемо отримати різні результати, то для кожного

випадку необхідно виконувати різні дії, а відповідно для користувачів демонструвати зміні результати.

Четвертим компонентом є база даних, яка відповідає за зберігання інформації про користувачів, платежів та інших необхідних даних.

П’ятий останній компонент – сервіс обробки цифрових платежів Stripe Api, що буде обробляти транзакції користувачів.

Визначивши основні структурні компоненти, необхідно об’єднати їх, надавши кожному роль та місце у додатку. Фронтенд об’єднується із базою даних та серверною частиною, щоб користувачі могли створити власний обліковий запис. Спершу запит від користувацької частини надходить до серверної, а потім серверна частина передає запит до бази даних та виконує необхідні функції з отриманою інформацією, за такою ж схемою відбувається доступ до усіх товарів, їх сортування за певними критеріями. Коли клієнт обрав певний товар і хоче оплатити його, то необхідно також використовувати сервіс обробки платежів, а також результати роботи протоколу 3D Secure.

На часовому проміжку можна зобразити процес наступним чином (рис. 2.1).

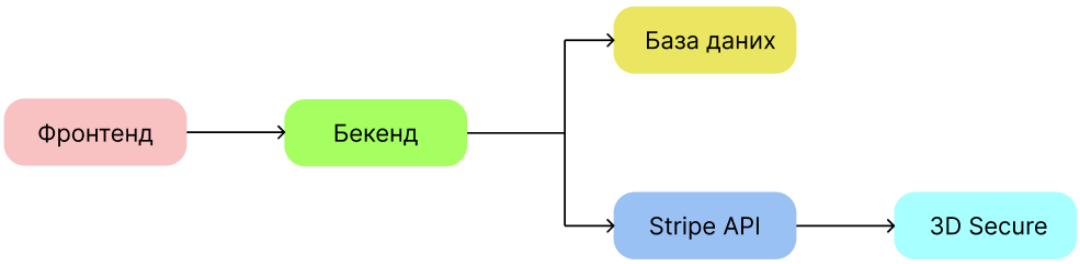


Рисунок 2.1 – Взаємодія компонентів

Якщо фронтенд, бекенд та база даних не вимагають додаткових пояснень через розповсюдженість, то як працює Stripe Api та 3D Secure разом варто розповісти.

Stripe Api – сервіс на базі Rest Api, що приймає закодовані запити та повертає відповідь у форматі JSON [20]. Щоб почати використовувати платформу, необхідно зареєструватися та отримати приватний та публічний

ключ, таким чином отримаєте доступ до Stripe Api та зможете використовувати його послуги у власному додатку.

Для розробки є корисна функція – тестовий режим, який не використовує реальні банківські дані, а лише спеціальні картки із заздалегідь визначеним результатом обробки карти. Якщо ж необхідно випустити сервіс для реальних користувачів, все що потрібно зробити це змінити приватний і публічний ключ доступу з тестових на реальні і сервіс буде повноцінно працювати із банками та обробляти дані транзакцій.

Публічний ключ – надається для фронтенд частини без ризику витоку даних, а приватний для безпечної передачі повідомлень між серверами.

Для реалізації оплати можна скористатися двома способами. Перший спосіб використовувати Stripe Checkout. Створений для тих, хто хоче отримати готове функціональне рішення, яке легко інтегрувати у власний магазин електронної комерції. Не вимагає знаних технічних знань чи необхідності підтримувати певну інфраструктуру. По суті – готове рішення для здійснення обробки платежів. Stripe Checkout надає готову форму, налаштовується під потреби користувача та дбає про безпеку платежів, потрібно лише мінімальні налаштування, як от вказати ~~ваші~~ банківські рахунки, куди надходитимуть кошти та під’єднати до власної розробки. Процес оплати завжди буде виглядати так:

– при оплаті замовлення, бекенд сервісу створює сесію Checkout і перенаправляє на сторінку оплати Stripe;

– користувач вводить необхідні дані для здійснення платежу;

– сервіс Stripe обробляє транзакцію та після успішного переказу повертає користувача на сторінку веб-сервісу, де було здійснено замовлення.

Тобто Checkout самостійно обробляє платежі за допомогою власного алгоритму, не потребуючи сторонньої участі.

Другим способом реалізації оплати є PaymentIntent. Даний сервіс не надає готових рішень, тож підходить тим, кому необхідно створити, унікальний процес оплати без пересилань на інші сторінки. Використовуючи PaymentIntent можливо керувати всім процесом оплати від авторизації до процесу обробки

платежу, тому вимагає сильних технічних знань, розуміння протоколів безпеки, для реалізації максимального рівня захищеності власного сервісу.

Процес оплати може варіюватися помітно в залежності від проекту, адже сервіс повністю поєднується з фронтенд частиною. Але один із можливих варіантів оплати може мати наступний вигляд:

– для ініціалізації оплати на бекенді запускається метод confirmCardPayment();

– якщо налаштовано функцію захисту від шахрайства 3D Secure, то в залежності від параметрів запуску, може з’явитись вікно підтвердження особи за допомогою різних методів;

– на основі результату попереднього пункту, авторизація або підтверджується або користувача просять пройти перевірку ще раз. Таким чином створюються дві сторінки, які будуть демонструватися користувачам у тому чи іншому випадку;

– коли авторизація пройдена обробляється платіж на основі алгоритму, який було задано для його здійснення.

– також можна співпрацювати із базами даних для зберігання деталей оплати, користувачів та товарів.

Даний спосіб дозволяє налаштувати повністю оплату під потреби користувача до механізмів безпеки та платежів. Але використання вимагає значних знань та наявності технічної команди для повноцінної і стабільної роботи додатку.

Порівнюючи Stripe Checkout та PaymentIntent можливо виділити ряд недоліків та переваг кожного з них. Перевагами Checkout є швидка інтеграція, легкість налаштування, автоматично запроваджує механізми захисту, завдяки чому підходить для більшості малих та середніх підприємств, яким потрібно легко і надійно запровадити можливість оплати на власній сторінці електронної комерції. Проте недоліком є неможливість впливати на користувацький інтерфейс оплати чи змінити певний механізм обробки транзакцій при необхідності.

Перевагою PaymentIntent є можливість створення нового інтерфейсу оплати та здійснювати повний контроль над процесом авторизації та оплати. Але така гнучкість породжує недолік у вигляді складного процесу інтеграції та налаштування правил. Відповідно потрібні технічні спеціалісти та відповідне обладнання для підтримки інфраструктури. Виходячи із перелічених переваг та недоліків, кожен власник обиратиме більш кращий варіант виходячи із вимог власного бізнесу.

Отже, розробка архітектури веб-сервісу є надзвичайно важливою для забезпечення стабільності, безпеки та ефективності системи. Архітектура визначає, як різні компоненти сервісу взаємодіють один з одним, як обробляються запити користувачів, як захищаються дані.

2.2 Розробка бази даних для веб-сервісу

Майже будь-який додаток у сучасному світі використовує бази даних, адже бізнесу необхідно керувати багатьма даними, причому забезпечуючи достатню швидкість та продуктивність, для того, щоб надати послуги, якомога більшій кількості клієнтів.

Веб-сервіс, що будемо розробляти, також потребує функціоналу баз даних, адже необхідно зберігати дані користувачів, товарів та покупок. Для успішного та інформативного користування додатком потрібно – розробити та спроектувати базу даних таким чином, щоб не виникало повторів, була достатня захищеність і швидкодія.

У роботі буде використовуватись нереляційна база даних MongoDB [21] – це потужна база даних, яка обробляє велику кількість інформації і забезпечує її постійний доступ.

Компанію заснували у 2007 році Дуайт Мерріман, Еліот Горовіц і Кевін Раян після успішного продажу рекламної онлайн-компанії Doubleclick, яку придбала Google. У ході роботи над Doubleclick автори швидко засвоїли обмеження реляційних баз даних, тому у 2007 році ґрунтуючись на своєму досвіді створили MongoDB. У той самий час хмарні обчислення з’являлися як нова бізнес-модель із обіцянками практично необмеженої масштабованості,

тарифікації за принципом оплати та відсутності необхідності самостійного керування IT-інфраструктурою. Мерріман і Горовіц побачили можливість скористатися перевагами цієї нової моделі для створення орієнтованої на розробника платформи [22].

Є декілька вагомих переваг сервісу перед традиційними реляційними базами даних.

Перша перевага полягає у тому, що не обов'язково спершу створювати чітку структуру схем, натомість використовується документо-орієнтована модель, що зберігає дані в JSON подібному форматі, тобто з часом схема може масштабуватись без необхідності створювати всю схему з самого початку. Особливо корисна властивість коли структурна схема постійно розвивається.

Друга перевага – це забезпечення максимальної продуктивності та надійності сервісу завдяки тому, що дані зберігаються у двійковому форматі BSON, що дуже ефективно для пошуку та роботи з даними. Також надаються додаткові функції, які допомагають відновити роботу після відмови та зберегти інформацію у тому вигляді, що була до відмови, завдяки механізму розподілу даних по вузлах. Під час функціонування у реальних умовах покращить показники доступності та надійності.

Третьою перевагою є обширна мова запитів, яка дозволяє швидко та зручно оперувати колекціями у спосіб, який вам необхідний.

До прикладу наведемо вигляд функцій, що дозволяють створювати, переглядати, оновлювати та видаляти об'єкти з бази даних.

Розглянемо операцію створення об'єкту:

db.sales.insertOne({ "_id" : 1, "item" : "abc", "price" : 10, "quantity" : 2, "date" : new Date("2014-03-01T08:00:00Z")});.

Використовуючи запит «db.sales» вказується сервісу, яку базу даних та колекцію використовувати, адже у одного користувача чи проекту може бути декілька колекцій і навіть баз. Запит «insertOne()» – безпосередньо застосовується для додавання одного запису у колекцію.

Операція перегляду:

```
db.sales.findOne( { "_id" : 1}, {"_id": 0 } );.
```

Знову спершу звертаємось до бази даних та колекції та здійснюємо пошук по заданим параметрам, що зберігаються у фігурних дужках.

Не менш важливо мати змогу редагувати дані, що виконується так:

```
db.employees.update({name: 'Ivan'},
{$set:{
name: 'Anatoly',
age: 50,
date_of_birth: '15.11.1972'}}).
```

Спочатку потрібно вказати об’єкт у якому відбуватимуться зміни, у прикладі це всі об’єкти властивість «name» який має значення «Ivan». Потім за допомогою «$set» встановлюються необхідні зміни у об’єкт.

Остання операція видалення, виконується наступним чином:

```
db.sales.deleteOne( {"_id":1} );
```

Функція «deleteOne()» видаляє більше непотрібну інформацію, використовуючи певні властивості об’єкта чи сукупність властивостей за необхідністю.

Щоб працювати з декількома записами необхідно замінити частину функції з «One()» на «Many()», дане правило справедливе для всіх розглянутих запитів.

Ще однією вагомою перевагою MongoDB велика спільнота користувачів, офіційна документація та велика кількість додаткових ресурсів для вирішення усіх питань, що виникатимуть у процесі розробки.

Проте, як і в будь-якому сервісі, існує ряд недоліків, які необхідно приймати до уваги, перш ніж брати технологію у розробку власного проекту. Оскільки MongoDB не притримується традиційно реляційної структури, то складні операції такі, як об’єднання чи транзакції не підтримуються. Також гнучкість у проектуванні схеми на початкових етапах може призвести до значних проблем у майбутньому, якщо були допущені помилки під час їх проектування. У свою чергу це викличе проблеми з продуктивністю та цілісністю даних. Варто зазначити, що використання MongoDB не є повністю безкоштовним, якщо

потрібно опрацьовувати велику кількість інформації та керувати одразу декількома базами даних, то даний набір можливостей доволі дороговартісний, особливо для малих та середніх організацій. Безкоштовна версія дозволяє працювати із однією базою даних, яка має певні обмеження по пам’яті.

Щоб перейти безпосередньо до проектування бази даних, необхідно визначити для чого будемо її використовувати, які дані повинні там зберігатися. Для успішного функціонування сервісу електронної комерції з інтеграцією платіжної платформи Stripe та використання протоколу 3D Secure потрібно зберігати дані про користувачів, товари та транзакції, які були здійснені.

Тож на основі методу «сутність-зв’язок», було прийнято рішення створити три сутності. Сутність «Користувачі» міститиме наступні атрибути:

– id;

– ім’я;

– пошта;

– роль;

– пароль.

Пошта та пароль необхідні для реалізації входу у персональний кабінет, а роль буде визначати, які дозволи матиме той чи інший користувач. Ролей планується дві – користувач та адміністратор. Адміністратор повинен мати можливість додавати нові товари та редагувати їх, і переглядати історію покупок користувачів. Окремий користувач повинен мати змогу лише здійснювати покупки, можливість додавати нові товари чи редагувати їх повинна бути недоступною.

Сутність «Товари» має містити наступні атрибути:

– id;

– назва;

– фото;

– кількість;

– ціна.

Товари мають обмежену кількість, тому важливо реалізувати зменшення їх кількості при кожній успішній оплаті. Атрибути назва, фото, ціна та рейтинг призначенні для того, щоб відображати їх потенційними покупцям.

Ще однією сутністю є «Транзакції», її атрибути:

– id;

– ім’я користувача;

– id користувача;

– ціна;

– статус транзакції.

– товар;

– дата.

У цій сутності повинні зберігатися усі спроби придбання товару, тобто інформація про користувача, а саме його ім’я та id для однозначної ідентифікації. Для того, щоб дозволити користувачу здійснювати транзакцію не лише за однин товар, а за декілька товарів, використаємо можливості MongoDB. Атрибут «товари» буде представлений як масив об'єктів, кожен об’єкт містить вибраний користувачем окремий товар і додаткове поле «кількість» для визначення кількості одиниць цього товару. Таким чином уникнемо створення додаткової сутності, яка б виконувала роль масиву об’єктів.

Оскільки для сутності «Транзакції» необхідно отримувати конфіденційні дані клієнтів з стороннього сервісу Stripe, використаємо спеціальні вебхуки від Stripe, які спершу налаштовується у персональному кабінеті Stripe так, як потрібно відповідно до завдання. У нашому випадку доцільно буде використати подію під час успішного та невдалого проведення платежу. Проте, перед тим, як зберігати отримані дані в розроблену базу даних, необхідно перевірити їх. Для перевірки використовуються спеціальні вебхуки, які потрібно порівнювати з власноруч обчисленим хешем. Якщо хеші збігаються це гарантує, що дані надійшли безпосередньо від сервісу Stripe та не були змінені у процесі передачі. Після успішного порівняння перевірені дані зберігаються у базі даних та використовуються користувачами. У разі невдалої перевірки, тобто хеш не збігається між собою, дані не будуть зберігатися у базі даних.

Зв’язки між сутностями представлено на рисунку 2.2.

Рисунок 2.2 – ER-модель розробленої бази даних

Зв’язок між сутностями «Користувачі» та «Транзакції» – 1:N, адже один користувач може здійснювати багато транзакцій. За таким ж принципом побудовано зв’язок між сутностями «Товари» та «Транзакції». Сутності «Користувачі» та «Товари» прямо непов’язані між собою, їх взаємодія відбувається через сутність «Транзакції».

Отже, проектування бази даних вимагає глибокого розуміння застосування, її задачі та вимоги. Також проектування бази даних є критично важливою частиною успішного функціонування інформаційних систем компанії. Ретельне планування та реалізація цієї структури дозволяє забезпечити високу продуктивність, надійність та безпеку збережених даних.

Визначили попередньою схему додатку, сутності та їх атрибути. Обґрунтували вибір бази даних, обговорили механізми забезпечення безпеки даних під час передавання через стороні сервіси та мережу Інтернет.

2.3 Розробка інтерфейсу для веб-сервісу

Від самого початку розвитку онлайн-сервісів значну кількість уваги приділяли покращенню користувацького досвіду. Адже приємний та зручний інтерфейс значно підвищують шанси того, що користувачі повернуться ще раз для отримання тих самих послуг та запросять своїх друзів до сервісу. У свою чергу це дозволить швидше розвиватись та отримувати прибуток. Тому даній частині веб-сервісу потрібно приділяти дуже багато уваги.

Під поняттям «хороший користувацький інтерфейс» розуміється не тільки загальна привабливість, а й зрозумілість попередніх та майбутніх дій. Тобто користувач повинен розуміти, який формат даних необхідно вводити у ту чи іншу форму. Якщо користувач ввів щось не правильно, він має розуміти, яку помилку від допустив. Також навігація по сервісу не повинна викликати зайвих запитань. Щоб реалізувати інтуїтивність користувацького інтерфейсу потрібно перед тим, як програмно реалізовувати програму, продумати весь процес, який користувач повинен пройти на сервісі та розподілити на логічні частини. Кожну окрему логічну частину відобразити на окремій сторінці.

Тож спершу користувач має авторизуватися, щоб отримати доступ до товарів та оплати, відповідно у кожного користувача має бути свій персональний кабінет. Сторінка товарів на якій будуть представлені всі товари. Для того, щоб здійснити покупку, користувач має знайти сторінку оплати та кошик, куди спершу додасть обрані товари.

Тобто основні сторінки веб-сервісу:

– сторінка реєстрації;

– сторінка входу;

– персональний кабінет;

– каталог товарів;

– кошик.

Окрім цього користувачі також, повинні мати доступ до історії транзакцій або своїх покупок та оцінювати товари.

Тож сторінки входу та реєстрації повинні містити форми вводу інформації та кнопки підтвердження (рисунок 2.2).

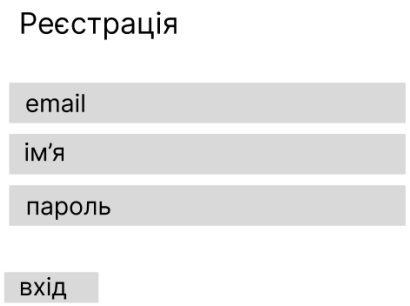


Рисунок 2.2 – Вигляд макету сторінки реєстрації

Введена інформація у поле «email» та «пароль» згодом буде використовуватись для ідентифікації користувача та надання йому прав при спробі входу. Сторінка входу при цьому буде мати схожий вигляд, лише буде відсутнє поле «ім'я».

Після успішного входження до системи користувач має бути переправлений на персональну сторінку. На власній сторінці повинна реалізована можливість переглядати історію здійснених покупок, а також переходити до каталогу товарів та кошику, представлено на рисунку 2.3.

Рисунок 2.3 – Вигляд макету сторінки персонального кабінету

Тож сторінка персонального кабінету буде основною звідки користувач матиме змогу отримати доступ до необхідних йому функцій.

Як бачимо з попередньої сторінки у нас є можливість потрапити ще на каталог товарів та кошик.

На сторінці каталогу товарів, маємо бачити безпосередньо товари, а також мати можливість додати обрані товари до кошику, сторінку зображено на рисунку 2.4.

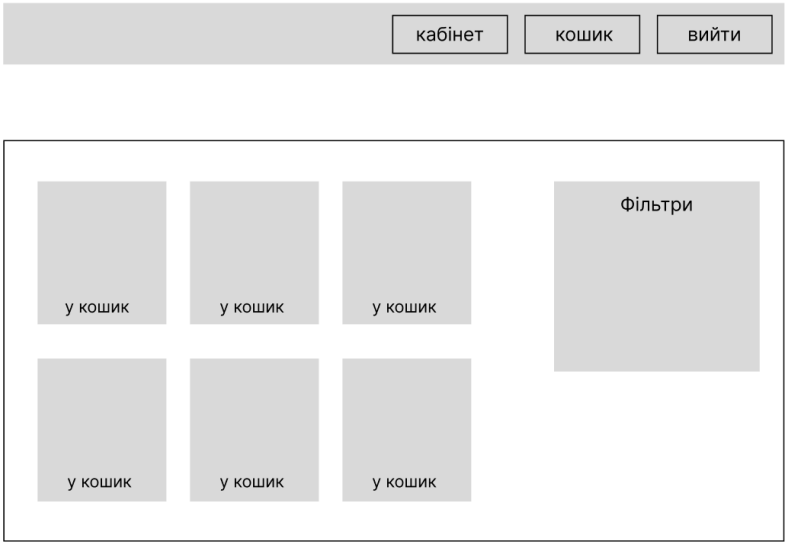


Рисунок 2.4 – Вигляд макету сторінки каталогу товарів

На даній сторінці користувач бачить усі доступні товари, може фільтрувати товари відповідно до своїх побажань.

Остання ключова сторінка це сторінка оплати на рисунку 2.5.

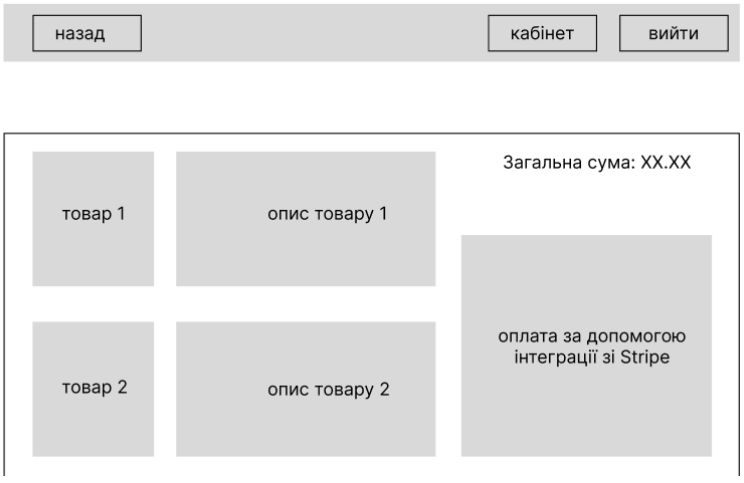


Рисунок 2.5 – Вигляд макету сторінки кошику

Обравши товар та перейшовши до кошику, покупець має бачити обрані товари, мати можливість взаємодіяти із ними збільшувати чи зменшувати кількість, а відповідно до цього і загальна сума має змінюватись. Оплата за допомогою Stripe буде включати у себе введення даних карти самостійно чи обрати інший спосіб оплати використовуючи інший сервіс. Якщо користувач

здійснив транзакцію, то куплені товари з’являться у персональному кабінеті разом з інформацією про статус платежу.

Отже, проектування інтерфейсу користувача також є невід’ємною частиною розробки будь-якого додатку. На даному етапі було розроблено макет, який слугуватиме основою при майбутній програмній розробці. Розроблений макет дозволяє візуалізувати головну структуру додатку, це допомогло виявити частину проблем ще на етапі проектування макету та вжити заходів для їх подолання. Також забезпечується цілісне уявлення та бачення кінцевої розробки.

2.4 Висновок до розділу 2

У цьому розділі, присвяченому дизайну інтерфейсу, розробці архітектури та бази даних, виконувався всебічний аналіз вимог і визначалися ключові компоненти системи. Відповідно до потреб користувача та функціональних характеристик програми, сутності та їхні атрибути розроблялися, щоб забезпечити зручний та ефективний спосіб зберігання даних.

Для реалізації сховища даних було обґрунтовано доцільність вибору сервісу бази даних MongoDB, яка пропонує високу продуктивність, масштабованість і гнучкість при обробці масивів даних, необхідних для зберігання інформації про продукти, користувачів і їхні транзакції.

Розроблено архітектуру програми, яка містить серверну частину для обробки запитів і клієнтську частину для взаємодії з користувачами, а також інтеграцію зі Stripe для забезпечення захищених онлайн-платежів.

Також приділялась увага механізмам безпеки, включаючи безпеку конфіденційних даних при обміні ними через сторонні сервіси та мережу Інтернет. Таким чином, створено міцну основу для реалізації системи, що відповідає вимогам як функціональності, так і безпеки, забезпечуючи зручність для користувачів та безпечну інтеграцію із зовнішніми сервісами.

3 РЕАЛІЗАЦІЯ ВЕБ-СЕРВІСУ ДЛЯ ЗАХИЩЕНИХ ІНТЕРНЕТ-ПЛАТЕЖІВ З ІНТЕГРАЦІЄЮ ПЛАТФОРМИ STRIPE ТА НАЛАШТУВАННЯМ МЕХАНІЗМУ ЗАХИСТУ 3D SECURE

У даному розділі здійснено вибір та обґрунтування мови програмування, бібліотек, крім того, розглянуто можливості програми із описом фрагментів коду, що відповідають за їх реалізацію. У результаті отримаємо додаток, який відповідає темі бакалаврської роботи.

3.1 Обґрунтування вибору мови програмування, бібліотек та середовища виконання

Розробивши архітектуру та функції сервісу, необхідно обрати мову програмування, бібліотеки та середовище виконання, для втілення проекту у повноцінний застосунок. Цей етап є дуже важливим, адже від цього залежить процес розробки та подальша підтримка застосунку.

Обрано мову програмування для програмної реалізації – JavaScript головною особливістю якої являється можливість зробити Web-сторінку інтерактивною, тобто такою, що реагує на дії користувача [23]. Також перевагою є те, що дана мова використовується для написання як користувацьких інтерфейсів, так і серверної частини. Недолік базується на динамічній типізації, через це зростає ймовірність помилки під час виконання програми, тому потребує більшої кількості часу на тестування та налагодження.

Оскільки мову програмування було обрано JavaScript, то для створення інтерфейсу користувача необхідно обирати серед популярних бібліотек та фреймворків, наприклад React.js, Angular чи Vue.js. Для того, щоб спростити та прискорити процес створення інтерактивного додатка і полегшити інтеграцію з серверною частиною ресурсу.

Angular – JavaScript фреймворк з відкритим кодом, створений у 2010 році компанією Google. Розробники класифікують Angular як повноцінну платформу розробки, яка включає в себе весь необхідний функціонал для розробки та не потребує підключення сторонніх інструментів [24]. Через свої особливості краще

підходить для написання великих, масштабованих і корпоративних проектів. Проте для невеликих – є надмірним, адже містить у собі велику кількість вбудованих інструментів, у результаті чого ускладнює початкове налаштування та потребує більшого часу на написання коду.

Vue JS – прогресивний JavaScript фреймворк з відкритим кодом, спрямований на розробку користувацьких інтерфейсів, що об'єднує в собі декілька ідей з бібліотеки React та фреймворку Angular, додаючи від себе нові можливості [24]. На відміну від Angular є досить швидким та простим у використані, але менша кількість спільноти, потенційно може призвести до неможливості знайти готові рішення та обговорення деяких аспектів.

React – бібліотека JavaScript, розроблена компанією Facebook. За час існування, технологія зібрала навколо себе багатомільйонну спільноту, завдяки чому вже існує багато користувацьких розширень, інструментів та готових рішень [24]. Оскільки React це бібліотека, а не фреймворк, то це дає суттєву перевагу, через більшу гнучкість у виборі інструментів для управління станом, маршрутизації тощо. Тобто при розробці є можливість самостійно обирати найбільш зручний інструмент, а не той, який представлений або підтримується у фреймворку. React як і Vue використовує Virtual DOM, що дозволяє змінювати лише ті елементи, які зазнали змін, а решту залишаючи без змін.

Для програмної реалізації поставленого завдання, де важлива гнучкість, швидкість роботи та простота інтеграції зі сторонніми сервісами, було обрано React. Велика спільнота дозволить швидко виконувати поставлені завдання, а готові рішення допоможуть покращити користувацький досвід завдяки загально прийнятим стандартам.

Середовищем виконання обрано Node.js – це серверна технологія, яка базується на мові програмування JavaScript та вийшла у 2009 році. Вона дозволяє розробникам використовувати JavaScript для створення серверних додатків та вебресурсів. Однак, на відміну від традиційного використання JavaScript, який працює на браузерах клієнта, Node.js виконує код на сервері [25]. Завдячуючи можливості працювати з асинхронним кодом та обробляти багато запитів одночасно, створюється масштабоване та швидке середовище для

розробки та підтримки додатків. Оскільки Node.js здобув широку популярність, то має активну спільноту розробників та велику кількість бібліотек та фреймворків, для прискорення роботи.

Отже, було обрано вибір мови програмування JavaScript та середовище виконання Node.js, проаналізовано популярні фреймворки та бібліотеки для розробки користувацького інтерфейсу на основі чого обґрунтовано вибір React.js.

3.2 Програмна реалізація веб-сервісу

У даному підрозділі демонструється інтеграція сервісу з Stripe API та 3D Secure, обробка сценаріїв з платежами, аутентифікація користувачів та налаштування додаткових політик.

Спершу створимо сервер у файлі «index.js», який буде обробляти та надсилати запити клієнтської частини і керуватиме логікою роботи усього додатку.

```
const express = require('express');
const dotenv = require('dotenv');
const cors = require('cors');
const cookieParser = require('cookie-parser');
const app = express();
app.use(cors({
   origin: 'http://localhost:5173',
   credentials: true}));
app.use(express.json());
app.use(cookieParser());
app.use(express.urlencoded({extended: false}))
require('dotenv').config();
const port = 8000;
app.listen(port, () => console.log(`Server is running on port ${port}`));
```

Використовуючи фреймворк Express, що буде керувати маршрутизацією у додатку, налаштовуємо базовий сервер, а також встановлюємо додаткові залежності для майбутньої роботи.

Перш ніж реалізувати можливість реєстрації і авторизації необхідно підключити базу даних. Для цього використаємо бібліотеку mongoose, яка з’єднує базу даних MongoDB та Node.js.

```
const { mongoose } = require('mongoose');
mongoose.connect(process.env.MONGO_URL)) {
```

```
then(() => console.log('database connected successfully!'))
.catch(() => console.log('error during connected to database'))
```

Для додаткової безпеки всі залежності винесемо у файл «.env».

Після підключення створимо форму для реєстрації у клієнтській частині у відповідному файлі «Register.jsx».

```
<form onSubmit={registerUser} className="form">
    <label >Name</label>
  <input type="text" placeholder='enter name' value={data.name} onChange={(e) =>
setData({ ...data, name: e.target.value })} />
    <label >Email</label>
    <input type="email" placeholder='enter email' value={data.email} onChange={(e) =>
setData({ ...data, email: e.target.value })} />
     <label >Password</label>
    <input type="password" placeholder='password' value={data.password} onChange={(e)
=> setData({ ...data, password: e.target.value })} />
     <button type='submit'>Submit</button>
 </form>
```

Та форму входу у файлі «Login.jsx».

```
    <form onSubmit={loginUser} className="form">
      <label>Email</label>
      <input type="email" placeholder='enter email' value={data.email} onChange={(e) =>
setData({ ...data, email: e.target.value })} />
      <label >Passworld</label>
      <input type="password" placeholder='password' value={data.password}
onChange={(e) => setData({ ...data, password: e.target.value })} />
      <button type='submit'>Submit</button>
    </form>
```

Підготувавши форми реалізуємо можливість реєстрації.

```
router.post('/register', registerUser);
const registerUser = async (req, res) => {
  console.log('Request body:', req.body);
  try {
    const { name, email, password, role } = req.body;
    const user = await User.create({name, email, password: hashedPassword, role: "User" }
    return res.json(user)
  } catch (error) { return res.json({error: error}) }}
```

Перш ніж занести нового користувача у базу даних перевіряємо чи вказав користувач своє ім’я, чи довжина паролю більше 8 символів та містить великі, малі літери і спеціальні символи. А також чи існує користувач з такою ж електронною поштою.

```
 if (!name) {
       return res.json({error: 'name is required'}) }
    if (!password || password.length < 8 || !/[a-z]/.test(password) ||
      !/[A-Z]/.test(password) ||
```

```
        !/[!@#$%^&*(),.?":{}|<>]/.test(password)
        return res.json({
          error: 'Password must be at least 8 characters long and include uppercase,
lowercase letters and a special character.' });}
      const exist = await User.findOne({ email });
      if (exist) {
        return res.json({error: 'email is taken alredy'})}
```

Також для безпечного зберігання паролю у базі даних, за допомогою bcrypt будемо зберігати у базі даних хеш паролю, а не сам пароль, що навіть при компрометації бази даних допоможе захистити облікові записи користувачів.

```
const hashPassword = (password) => {
  return new Promise((resolve, reject) => {
    bcrypt.genSalt(12, (err, salt) => {
      if (err) { reject(err); }
      bcrypt.hash(password, salt, (err, hash) => {
        if(err) { reject(err)
         resolve(hash) })});});}
const hashedPassword = await hashPassword(password)
    const user = await User.create({
      name, email, password: hashedPassword, role: "User"})
    return res.json(user)
```

Відповідно при аутентифікації користувачів будемо порівнювати хеш веденого паролю із збереженим у базі даних, якщо вони однакові – то паролі збігаються.

```
router.post('/login', loginUser);
 const loginUser = async (req, res) => {
  try {
    const { email, password } = req.body
    const user = await User.findOne({ email });
    if (!user) {
      return res.json({error: 'User dosen`t exist' })}
    const match = await comparePassword(password, user.password)
    if (match) {
      jwt.sign({ email: user.email, id: user._id, name: user.name, role: user.role },
process.env.JWT_SECRET, {}, (err, token) => { res.cookie('token', token).json(user) })
    if (!match) {  res.json(
{  error: 'Password do not match'}) }
```

Окрім паролю перевіряємо чи збігаються ім’я користувача. Якщо інформація однакова, то генеруємо токен jwt, завдяки якому користувачі будуть отримувати доступ до захищеного вузла. Тож даний функціонал забезпечує безпечну аутентифікацію користувачів, завдяки сучасним рішенням.

Далі потрібно реалізувати можливість додавати, видаляти та оновлювати товари для адміністраторів. Спершу створимо форму для додавання товарів.

```
<form onSubmit={AddProduct} className="form add">
  <label >Name Product</label>
   <input type="text" placeholder='enter name product' value={data.name} onChange={(e) => setData({ ...data, name: e.target.value })} />
  <label> Description</label>
   <input type="text" placeholder='enter description product' value={data.decript} onChange={(e) => setData({ ...data, decript: e.target.value })} />
  <label >Price</label>
   <input type="number" value={data.price} onChange={(e) => setData({ ...data, price: e.target.value })} />
  <label >Quantity</label>
  <input type="number" value={data.cuantity} onChange={(e) => setData({ ...data, cuantity: e.target.value })} />
  <label >Images URL</label>
 <input type="text" value={data.img} onChange={(e) => setData({...data, img: e.target.value })} placeholder='Choose Images URL' />
  <button type='submit'>Submit</button>
</form>
```

Після створення форми, необхідно створити функцію на сервері, яка буде перевіряти введені дані користувача та додавати до бази даних у разі правильності ведичних даних.

```
router.post('/add', addProduct);
const addProduct = async (req, res) => {
  try {
    const { name, decript, img, price, cuantity } = req.body;
    if (!name || !decript || !img) {
      return res.json({error: 'all filds is required'  })
    const exist = await Product.findOne({ name });
    if (exist) {
      return res.json({ error: 'Name is shoud be unique'})
    const product = await Product.create({ name, decript, img, price, cuantity }
    return res.json(product)
  } catch (error) {
    return res.json({error: error)}
```

Для інших функцій таких, як оновлення чи видалення аналогічно створюється форма та обробник на сервері.

Після додавання товарів у базу даних користувачі мають побачити їх та мати можливість додати обраний товар до кошика. Для цього додамо всі товари на головну сторінку користувачів.

```
const getProducts = async (req, res) => {
  try {
    const products = await Product.find();
    res.json(products);
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch products' });
```

Попередньо створивши сторінку для відображення отриманих товарів.

```
export default function WorkPanel() {
const [card, setCard] = useState([]);
   const AddToCard = (product) => {
      const IsAlreadyInCard = card.some(p => p._id === product._id)
      if (!IsAlreadyInCard) {
         setCard(prev => [...prev, product]);
         toast.success("Product added in card")
         totalPrice()
      } else {
         toast.error("Product is alredy in card") }}
   return (
      <button onClick={handleCard}>Cart</button>
      {products?.length > 0 && products.map(product => (
            <div key={product._id} className="product">
             <img src={product.img} className="product__img" />
              <div className="product__name">{product.name}</div>
               <div className="product__price">${product.price}</div>
              <p className="product__descript">{product.decript}</p>
                 {product.cuantity === 0 ? <button disabled >Out of stock</button> : <button
onClick={() => AddToCard(product)}>Buy</button>}
```

З моменту коли товари додано у кошик, то користувач може оплатити доданий товар, тому на цьому етапі потрібно інтегрувати Stripe, щоб надати таку можливість.

```
const Stripe = require('stripe')(process.env.STRIPE_SECRET_KEY)
 const paymentIntent = await Stripe.paymentIntents.create({
        amount: totalAmount * 100,
        currency: 'usd',
        payment_method: stripeToken,
        confirm: true,
        return_url: 'http://localhost:5173/card',
        automatic_payment_methods: {
          enabled: true,
          allow_redirects: 'never'}});
```

Далі створимо користувацький інтерфейс для оплати замовлення використовуючи вбудовану форму вводу платіжних даних Stripe.

```
const stripe = useStripe();
const elements = useElements();
const cardElement = elements.getElement(CardElement);
 <div className="card">
          {card.map(item => (
            <div key={item._id} className="card__item">
              <p className="card__name">{item.name}</p>
     <p className="card__price">Price: ${item.price}</
<button onClick={() => removeProduct(item._id)}>Remove</button>
            </div>
   <h3>Загальна сума: ${total}</h3>
```

```
<div className="cardElement">
        <CardElement options={CARD_OPTIONS}/>
      </div>
```

Таким чином створено сторінку кошика, де зазначено товари, які користувач додав, загальна сума та форма ведення платіжних даних від Stripe.

Ввівши інформацію у спеціальні поля, необхідно підготувати логіку обробки платежу.

```
try {
const attempt = await Transaction.create({
        name_user: user.name,
        userId: user._id,
        price: totalAmount,
        product: productIds.map(product => product._id),
        status: paymentIntent.status
    });
    if (paymentIntent.status === "requires_action") {
      return res.json({
        requires_action: true,
        client_secret: paymentIntent.client_secret});}
if (paymentIntent.status === "succeeded") {
        await Promise.all(
          card.map(item => Product.findByIdAndUpdate(item._id, {
              $inc: { cuantity: -1 }}))}
catch (error) {  res.status(500).json({ error: error.message });
```

Спершу надсилається запит до Stripe API з платіжними даними клієнта й отримавши відповідь від сервера оброблюємо її. Якщо спроба успішна – зменшимо кількість наявного товару. Якщо платіж має статус «requires_action», тобто потребує додаткових дій від користувача, то вимагатимемо проходження 3D Secure від користувача при умові, що його картка підтримує даний стандарт.

Додатково налаштуємо список країн, для яких обов’язково потрібно вимагати проходження 3D Secure, якщо картка підтримує цю функцію.

```
const highRiskCountries = ['MY', 'MX', 'IN', 'TH'];
const paymentMethod = await Stripe.paymentMethods.retrieve(stripeToken);
const cardCountry = paymentMethod.card.country;
const requireThreeDSecure = highRiskCountries.includes(cardCountry);
const paymentIntent = await Stripe.paymentIntents.create({
        amount: totalAmount * 100,
        currency: 'usd',
        payment_method: stripeToken,
        confirm: true,
        return_url: 'http://localhost:5173/card',
        payment_method_options: {
          card: { request_three_d_secure: requireThreeDSecure ? 'any' : 'automatic'}},
```

```
        automatic_payment_methods: {
          enabled: true,
          allow_redirects: 'never'}});
```

Таким чином, забезпечується зниження ризику втрати коштів для бізнесу та користувачів.

Ще однією політикою захисту від підбору платіжних даних та захисту сервера від перевантаження є обмеження на кількість транзакцій у часі.

```
const totalAmount = card.reduce((acc, item) => acc + item.price, 0);
    const lastAttempts = await Transaction.find({
      userId: userId,
      createdAt: { $gte: new Date(Date.now() - 5 * 60 * 1000) }});
    if (lastAttempts.length >= 3) {
      return res.status(429).json({ error: "Too many attempts already tried. Please try again later." }); }
```

Даний код реалізує обмеження на здійснення більше трьох успішних транзакцій для одного користувача протягом п’яти хвилин. Це значно ускладнює проведення автоматизованих атак та дає час для реагування системам захисту.

Останньою політикою є запровадження денного ліміту для користувачів. Тобто встановлення суми більше, якої не можливо придбати продукт за один день користувачеві.

```
const dailyTransactions = await Transaction.find({
      userId: userId,
      status: "succeeded",
      createdAt: { $gte: new Date(Date.now() - 24 * 60 * 60 * 1000) } });
    const dailyTotal = dailyTransactions.reduce((sum, tx) => sum + tx.price, 0);
    const DAILY_LIMIT = 1500;
    if (dailyTotal + totalAmount > DAILY_LIMIT) {
      return res.status(403).json({
        error: `The daily limit of ${DAILY_LIMIT} has been exceeded. Today, we have already spent ${dailyTotal}`});}
```

Ліміт у розробленому додатку встановлено на рівні 1500 одиниць. У разі компрометації облікового запису запобігає значним фінансовим втратам та надає час користувачам зреагувати та вжити заходів для відновлення доступу до свого облікового запису.

Отже, сервіс забезпечує надійну аутентифікацію, захист платіжних операцій, а розроблені додаткові політики запобігають автоматизованим атакам, знижують ризик шахрайських транзакцій та забезпечують стабільну роботу системи.

3.3 Тестування розробленого додатку

Підрозділ присвячений проведенню тестування розробленого веб-сервісу. Оцінимо відгук додатка на різні користувацькі сценарії, щоб зрозуміти чи відповідає сервіс встановленим вимогам.

Перша сторінка, яка зустрічає користувача, містить ознайомчу інформацію та посилання на проходження реєстрації або входу на сайт зареєстрованим користувачам (рис. 3.1).

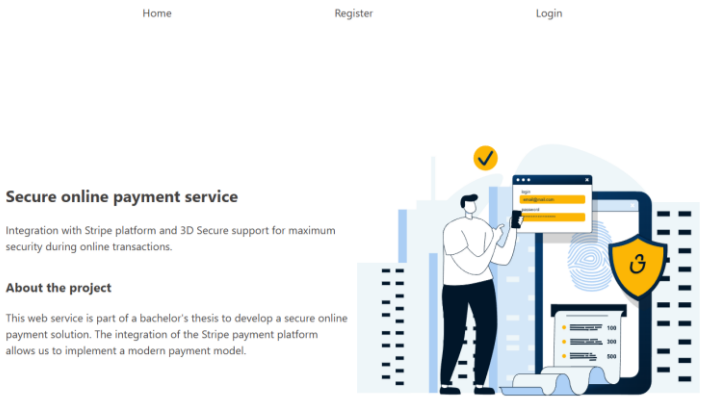


Рисунок 3.1 – Початкова сторінка сервісу

Перейшовши на сторінку «Register» користувач може зареєструватися. Користувачу необхідно правильно заповнити форму реєстрації, інакше сервіс не зареєструє нового користувача. Першим перевіряється чи заповним користувач ім’я, якщо користувач залишив дане поле порожнім – з’явиться відповідне повідомлення, як на рисунку 3.2.

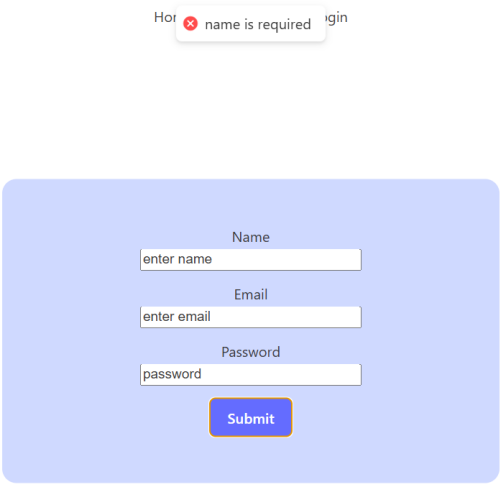


Рисунок 3.2 – Вигляд вікна при незаповненому полю «Name»

Схожим чином відбувається повідомлення про невідповідність паролю поставленим умовам (рис. 3.3).

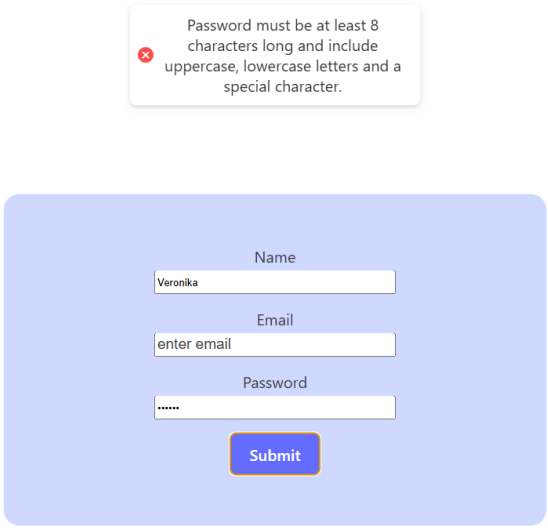


Рисунок 3.3 – Результат перевірки паролю

Введений пароль «123456» не відповідає жодній із поставлених вимог, тому виводиться попередження.

На останок перевіряється чи користувач ввів унікальну електрону пошту, якщо ж обліковий запис з такою поштую існує отримаємо попередження (рис. 3.4)

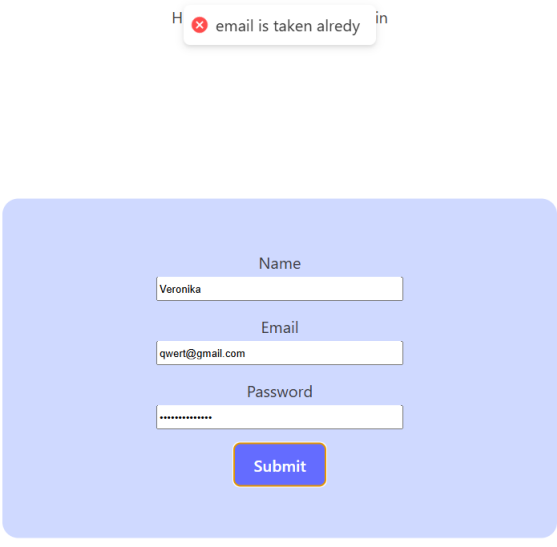


Рисунок 3.4 – Вигляд вікна сервісу, якщо обліковий запис з таким іменем існує

При правильно заповнених полях, новий користувач отримає повідомлення, що продемонстровано на рисунку 3.5.

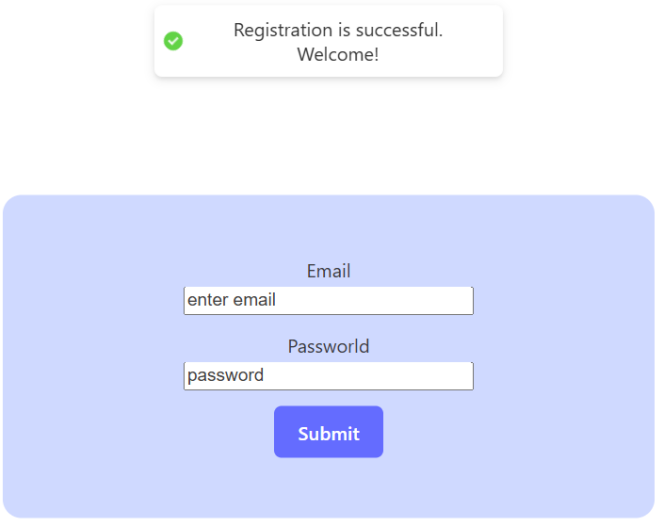


Рисунок 3.5 – Повідомлення про успішну реєстрацію

Одразу після підтвердження реєстрації, користувача автоматично переспрямовують на сторінку входу.

Після входу в обліковий запис користувач побачить інтерфейс, подібний до зображеного на рисунку 3.6

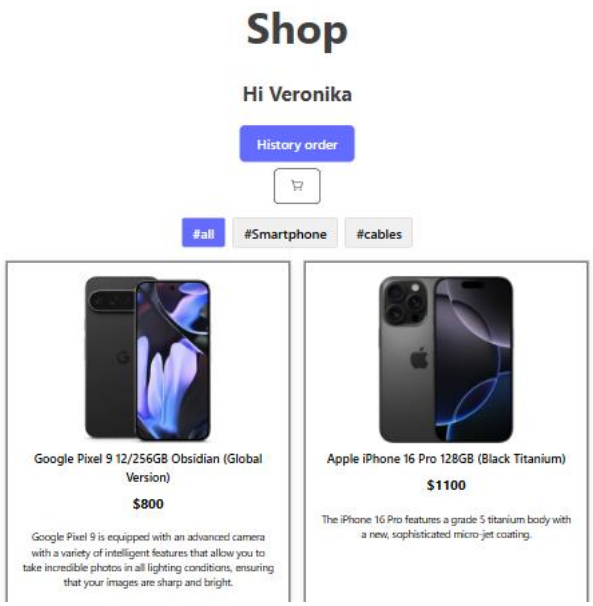


Рисунок 3.6 – Вигляд сторінки облікового запису користувача

З головної сторінки можливо перейти на сторінку історії замовлень, сторінку електронного кошику та обрати товар для замовлення, використовуючи фільтри.

Спробуємо зайти на сторінку облікового запису без попередньої авторизації, результат наведений на рисунку 3.7.

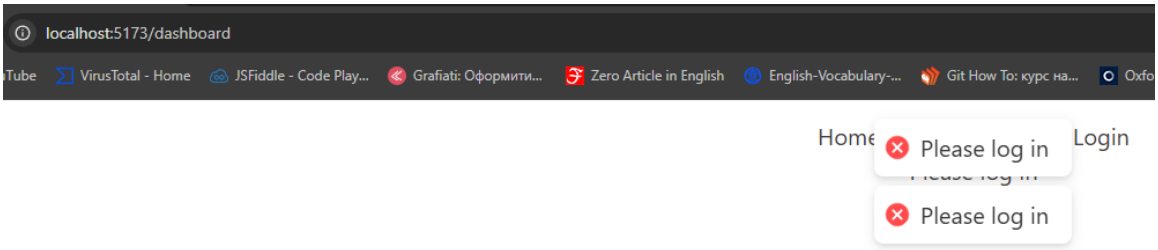


Рисунок 3.7 – Вигляд вікна при неавторизованій спробі входу у особистий кабінет

Тож неавторизованим користувачам не вийде увійти до сервісу.

Додавши товари у кошик перейдемо до перевірки сценаріїв оплати. Для тестування різних ситуацій, наприклад на картці не вистачає коштів, потрібні додаткові дії від користувача, тощо будемо використовувати номери тестових карток від сервісу Stripe Api [26]. У свою чергу Stripe гарантує, що логіка обробки платежів у тестовому режимі повністю відповідає реальним умовам проведення платежу, тобто сервіс не просто імітує відповідь на конкретну картку, а моделює процес обробки реальних платіжних даних. Завдяки цьому є можливість для розробників безпечно тестувати надійність розробленого сервісу, а потім легко перейти до реальних платежів.

Наприклад картка 4242 4242 4242 4242 не потребує додаткового проходження 3D Secure та має успішно завершити транзакцію (рисю 3.8).

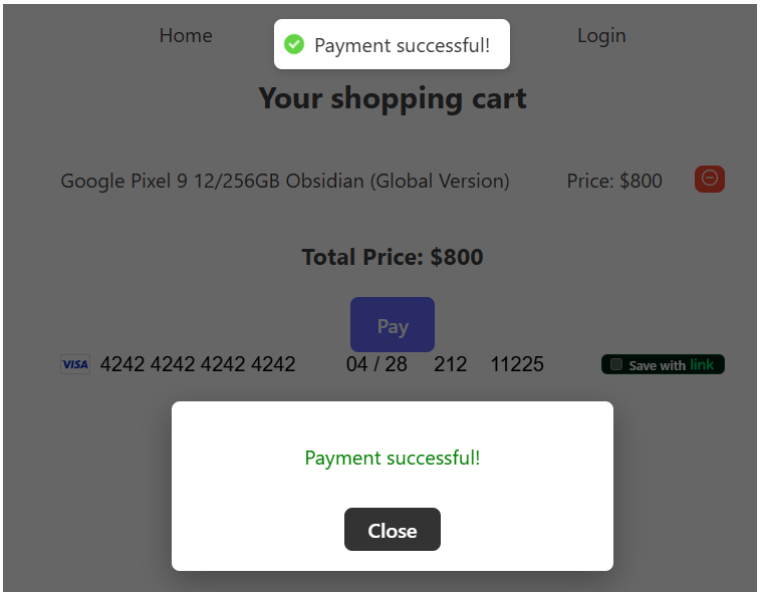


Рисунок 3.8 – Вигляд вікна при успішному завершені платежу

На рисунку 3.8 для сплати товару обрано картку 4242 4242 4242 4242, сума замовлення складає 800. Щоб перевірити чи дійсно відбувався успішний платіж перейдемо на сторінку істрії замовлень (рис. 3.9).

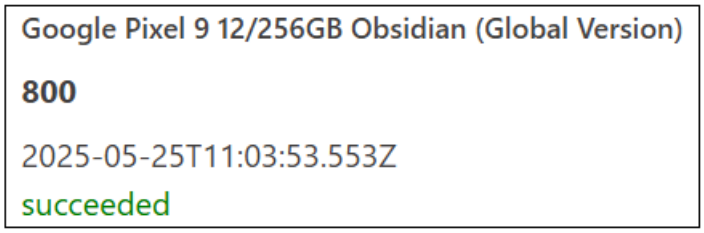


Рисунок 3.9 – Запис про успішно пройдене замовлення

Як бачимо із рисунка 3.9 – платіж дійсно відбувся з успішним статусом. Далі оберемо картку 4000000000009995, яка призведе до відхилення платежу через недостатню кількість коштів на рахунку (рис. 3.10).

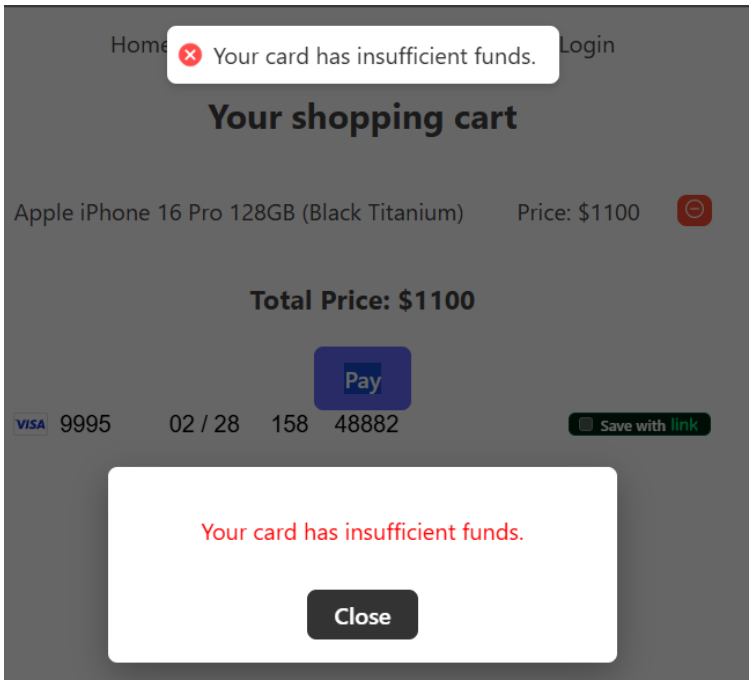


Рисунок 3.10 – Відхилення платежу через недостатню кількість коштів на рахунку

Проаналізувавши рисунки 3.9 та 3.10 можемо зробити висновок, що сервіс успішно обробляє платежі.

Перейдемо до тестування політики денного обмеження на певну суму. Після успішного завершення платежу на рисунку 3.9, спробуємо створити замовлення так, щоб утворена сума перевищувала денний ліміт (рис. 3.11).

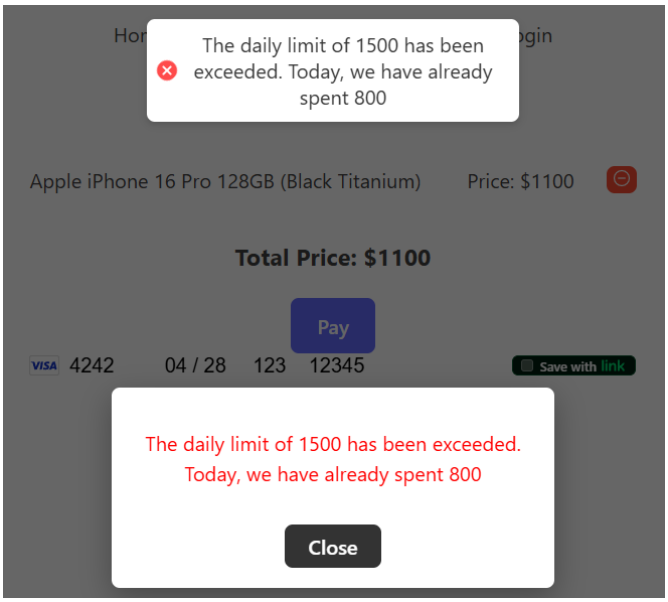


Рисунок 3.11 – Вигляд вікна при перевищені денного ліміту

Користувач бажає створити ще одне нове замовлення на суму 1100, але його попереднє замовлення коштувало 800, тож разом утворена сума перевищує ліміт.

Спробуємо перевірити логіку сервісу на обмеження кількості спроб за п’ять хвилин. Створимо успішні три замовлення, але четвертий має не відбутися (рис. 3.12).

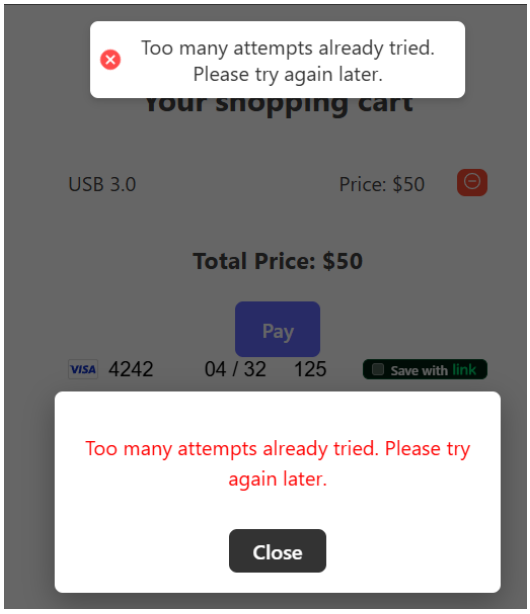


Рисунок 3.12 – Вигляд вікна при четвертій спробі здійснити платіж за короткий проміжок часу

Як видно з рисунку 3.12, дійсно виникає повідомлення про обмеження, але щоб перевірити чи справді не відбувається платіж перейдемо на сторінку історії замовлень (рис. 3.13).

Google Pixel 9 12/256GB Obsidian (Global Version) **800** 2025-05-25T11:03:53.553Z succeeded	USB 3.0 **50** 2025-05-25T11:35:12.555Z succeeded	USB 3.0 **50** 2025-05-25T11:35:16.457Z succeeded	USB 3.0 **50** 2025-05-25T11:35:19.987Z succeeded

Рисунок 3.13 – Вигляд сторінки історії замовлень покупця

Проаналізувавши рисунок 3.13 бачимо, що відбулось три платежі за короткий проміжок часу, але четвертого немає, отже додаток правильно обробляє дані правила.

Щоб перейти до останнього правила, тобто вимагати обов'язкове проходження 3D Secure, якщо картка зареєстрована у певній країні, потрібно протестувати, як виконується обробка необхідності проходження 3D Secure від користувача.

Для цього скористаємось карткою 4000002760003184, яка завжди вимагає проходження 3D Secure (рис. 3.14).

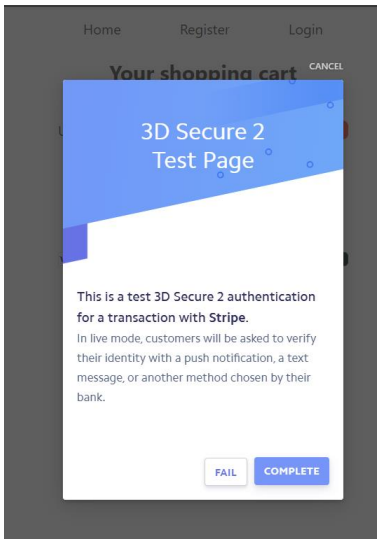


Рисунок 3.14 – Проходження 3D Secure

Під час оплати з'являється вікно, що зображено на рисунку 3.14, яка імітує успішний, якщо натиснути «Complete» чи неуспішний «Fail» процес додаткової аутентифікації.

Тож протестуємо політику обов'язкового проходження 3D Secure для певних країн (рис. 3.16). Скористаємось картками 4000003560000008, що належить країні – Індія та 4000000360000006 – Австралія. Індія – у розробленому додатку позначена, як країна з високим рівнем ризику, тому потрібно вимагати додаткової аутентифікації.

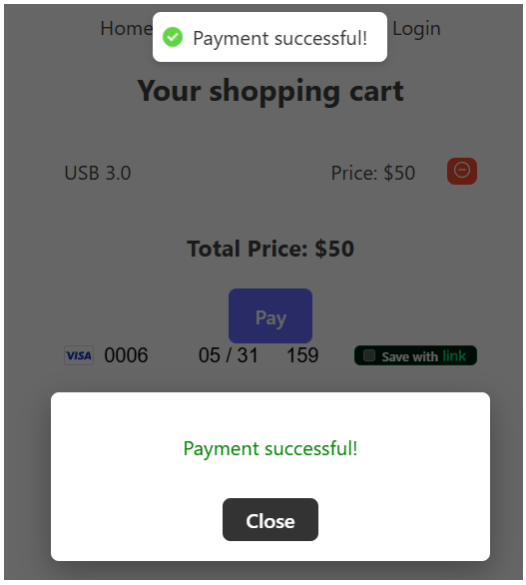


Рисунок 3.16 – Результат перевірки картки 4000000360000006(Австралія)

Як бачимо з рисунку 3.16, платіж одразу підтвердився не потребуючи від користувача проходження 3D Secure.

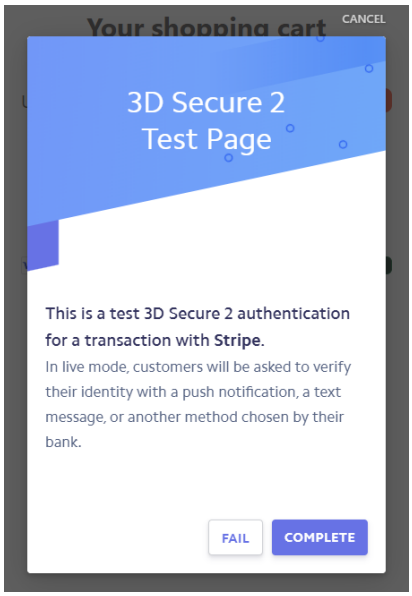


Рисунок 3.17 - Результат перевірки картки 4000003560000008 (Індія)

З рисунку 3.17 бачимо, що потрібно проходити додаткову аутентифікацію перед підтвердженням платежу.

Отже, розроблений додаток відповідає усім поставленим вимогам, виконує розроблені функції належним чином і демонструє стабільну роботу та безпечну роботу у тестовому середовищі.

3.4 Висновок до розділу 3

У даному розділі було здійснено практичну реалізацію сервісу для захищених інтернет-платежів з інтеграцією платформи Stripe та налаштуванням механізму захисту 3D Secure. Перед практичним виконанням було обрано мову програмування JavaScript, бібліотеку React та фреймворк Express.js для виконання поставлених цілей.

Було проведено тестування різних користувацьких сценаріїв, щоб впевнитись, що додаток відповідає поставленим вимогам та безпечно обробляє події. Зокрема модульовано різні платіжні сценарії, завдяки офіційній документації Stripe. Результати тестування засвідчили коректну роботу системи, адже усі реалізовані сценарії працювали надійно та очікувано.

У результаті роботи отримано надійний сервіс для обробки платежів, який у майбутньому легко підтримувати та масштабувати за потреби.

ВИСНОВКИ

У процесі виконання бакалаврської роботи було реалізовано веб-сервіс для безпечного проведення онлайн-платежів із використанням платформи Stripe та впровадженням додаткового механізму захисту 3D Secure.

Основною метою дослідження було створення захищеного середовища для обробки онлайн-платежів, що відповідатиме сучасним вимогам до безпеки платіжних сервісів.

У ході роботи було:

– проаналізовано сучасні платіжні сервіси, їх переваги та недоліки, особливості забезпечення безпеки у кожному з них;

– досліджено принцип роботи технології 3D Secure, необхідність розвитку різних версій даного стандарту, їх недоліки і переваги. Вплив версії на користувацький досвід та збільшення продажів;

– спроектовано архітектуру веб-сервісу, що включає клієнтську частину – для покращеної взаємодії з користувачем, серверну частину – для безпечної обробки інформації та базу даних – що забезпечує надійне зберігання;

– реалізовано інтеграцію з платформою Stripe, у тому числі із механізмом 3D Secure та налаштуванням додаткових політик безпеки;

– забезпечено збереження статусів платежів у базі даних з урахуванням можливих сценаріїв: успішна оплата, помилка, потреба в додаткових діях, тобто проходження 3D Secure;

– проведено тестування сервісу в тестовому середовищі з використанням тестових карт, яке підтвердило, що поведінка системи стабільна та передбачувана;

Практичне значення виконаної роботи полягає в можливості використання розробленого сервісу, як основи для створення повноцінного онлайн-магазину або іншого застосунку з необхідним функціоналом захищених оплат.

Завдяки застосуванню сучасних технологій (Stripe API, 3D Secure) забезпечено високу надійність та адаптивність рішення до реальних потреб бізнесу.

ПЕРЕЛІК ПОСИЛАНЬ

1. 43 ECommerce Statistics In 2024 (Global And U.S. Data) - SellersCommerce. URL: https://www.sellerscommerce.com/blog/ecommerce-statistics/ (дата звернення: 22.03.2025).

2. Офіційний сайт PayPal. URL: https://www.paypal.com/ua/home (дата звернення: 22.03.2025).

3. Офіційний сайт Braintree URL: https://www.braintreepayments.com/ (дата звернення: 23.13.2025).

4. Офіційна сторінка ціноутворення сервісу онлайн-платежів Braintree URL: https://www.braintreepayments.com/braintree-pricing (дата звернення: 23.03.2025).

5. Офіційний сайт 2Checkout URL: https://www.2checkout.com/ (дата звернення: 24.03.2025).

6. Офіційний сайт Adyen. URL: https://www.adyen.com/ (дата звернення: 24.03.2025).

7. Get started | adyen docs. URL: https://docs.adyen.com/payouts/payout-service/get-started/ (дата звернення: 24.03.2025).

8. Офіційний сайт Stripe. URL: https://stripe.com/ (дата звернення: 25.03.2025).

9. What is 3D secure?. URL: https://www.verifi.com/chargebacks-disputes-faq/what-is-3d-secure/ (дата звернення: 28.03.2025).

10. Салієва О. В., Порицька В. В. Порівняння існуючих версій протоколу 3-d secure. м. Вінниця. 2025. URL: https://conf.vntu.edu.ua/index.php/all-fm/all-fm-2025/paper/view/23299/19281 (дата звернення: 29.03.2025).

11. Frictionless authentication with 3D secure 2. URL: https://3dsecure2.com/frictionless-flow/ (дата звернення: 02.04.2025).

12. Syarif A., Satria M.H., Gabteni H. Tailoring data storage configuration for efficient fraud detection model training. Jurnal Ilmiah Teknik Elektro Komputer dan Informatika.2024. – Т. 10, № 4. – с. 1021–1036.

URL: https://journal.uad.ac.id/index.php/JITEKI/article/view/30013 (дата звернення: 04.04.2025).

13. 3D secure versions explained. URL: https://solidgate.com/blog/3d-secure-versions-explained/#How_Does_3D_Secure_Work_To_Enhance_Online_Payment_Security (дата звернення: 05.04.2025).

14. Ще безпечніші платежі: EMV 3-D Secure в НПС “ПРОСТІР”. URL: https://bank.gov.ua/ua/news/all/sche-bezpechnishi-plateji-emv-3-d-secure-v-nps-prostir (дата звернення: 05.04.2025).

15. Harmonizing open banking in the European Union: an analysis of PSD2 compliance and interrelation with cybersecurity frameworks and standards / M. Gounari та ін. *International cybersecurity law review*. 2024. URL: https://doi.org/10.1365/s43439-023-00108-8 (дата звернення: 06.04.2025).

16. Payment processor vs. payment gateway: what they are and how they work together. URL: https://stripe.com/resources/more/payment-processor-vs-payment-gateway (дата звернення: 07.04.2025).

17. The definitive guide to PCI DSS version 4 documentation, compliance, and management / B. Cooper Jr та ін. 2023. 88 с.

18. Paul Voigt, Axel von dem Bussche. The EU general data protection regulation (GDPR). 385 с.

19. PSD2 - second payment services directive. Information Set. URL: https://www.financelatvia.eu/wp-content/uploads/2018/04/PSD2-Second-Payment-Services-Directive-.pdf (дата звернення: 10.04.2025).

20. Stripe API reference. URL: https://docs.stripe.com/api (дата звернення: 11.04.2025).

21. Офіційна сторінка MongoDB. URL: https://www.mongodb.com/ (дата звернення: 12.04.2025).

22. Our story. MongoDB. URL: https://www.mongodb.com/company/our-story (дата звернення: 12.04.2025).

23. Киселевич В., Вербівський Д. Аналіз мови програмування JavaScript. Тези доповідей І Всеукраїнської науково-технічної конференції

«Комп’ютерні технології: інновації, проблеми, рішення», м. Житомир, 20 жовтня 2018 р. URL: https://conf.ztu.edu.ua/wp-content/uploads/2019/02/31-1.pdf (дата звернення: 06.05.2025).

24. Киселевич В., Вербівський Д. Огляд технологій для створення односторінкових вебдодатків. IV Всеукраїнська науково-практична конференція "Сучасні інформаційні технології в освіті та науці", 8 листопада 2019 р. URL: http://eprints.zu.edu.ua/35620/1/збірник_22-247-251%20(1).pdf (дата звернення: 06.05.2025).

25. Використання Node.js у веброзробках: переваги технології у рішеннях від Softline. URL: https://softline.company/ua/news/vykorystannia-nodejs-u-vebrozrobkakh-perevahy-tekhnolohii-u-rishenniakh-vid-softline.html (дата звернення: 15.05.2025).

26. Test card numbers. Stripe Docs. URL: https://docs.stripe.com/testing (дата звернення: 18.05.2025).

ДОДАТКИ

Вінницький національний технічний університет
Факультет менеджменту та інформаційної безпеки
Кафедра менеджменту та безпеки інформаційних систем

ЗАТВЕРДЖУЮ
Голова секції "Управління інформаційною безпекою" кафедри МБІС
д.т.н., професор
_____ Юрій ЯРЕМЧУК
" 20 " березня 2025 р.

ТЕХНІЧНЕ ЗАВДАННЯ

до бакалаврської кваліфікаційної роботи на тему:

«Розробка веб-сервісу для захищених інтернет-платежів з інтеграцією платформи Stripe та налаштуванням механізму захисту 3D Secure»

08-72.БКР.021.00.000 ТЗ

Керівник бакалаврської кваліфікаційної роботи
к.т.н., доцент
_____ Салієва О.В.
"____" _____

Вінниця – 2025 р

Додаток А. Технічне завдання

1. Найменування та область застосування

Веб-сервіс для захищених інтернет-платежів з інтеграцією платформи Stripe та налаштуванням механізму захисту 3D Secure. Область застосування: захист онлайн-платежів.

2.Підстава для розробки

Розробка виконується на основі наказу ректора ВНТУ № 97 від 20.03.2025 р.

3. Мета та призначення розробки

3.1 Мета розробки: розробка веб-сервісу для безпечних інтернет-платежів, інтегруючи платіжну платформу Stripe та налаштовуючи механізм захисту 3D Secure.

3.2 Призначення: розроблений програмний засіб призначений для проведення захищених електронних платежів.

4. Джерела розробки

4.1 Офіційний сайт Stripe. URL: https://stripe.com/ (дата звернення: 23.10.2024).

4.2 Салієва О. В., Порицька В. В. Порівняння існуючих версій протоколу 3-d secure. м. Вінниця. 2025. URL: https://conf.vntu.edu.ua/index.php/all-fm/all-fm-2025/paper/view/23299/19281 (дата звернення: 06.05.2025).

4.3 Paul Voigt, Axel von dem Bussche. The EU general data protection regulation (GDPR). 385 с.

4.4 The definitive guide to PCI DSS version 4 documentation, compliance, and management / B. Cooper Jr та ін. 2023. 88 с.

5. Вимоги до програми

5.1 Вимоги до функціональних характеристик:

5.1.1 Програмний засіб повинен мати зручний, легкий у використанні інтерфейс користувача;

5.1.2 Реалізація методу не повинна вимагати спеціальних ліцензійних програмних додатків;

5.1.3 Програмний засіб повинен виконувати процес автентифікації користувачів у системі.

5.2 Вимоги до надійності:

5.2.1 Програмний засіб повинен працювати без помилок, у випадку виникнення критичних ситуацій необхідно передбачити виведення відповідних повідомлень;

5.2.2 Бази даних повинні бути налаштовані на автоматичне створення резервних копій;

5.2.3 Програмний засіб повинен виконувати свої функції.

5.3 Вимоги до складу і параметрів технічних засобів:

– процесор – Pentium 1500 МГц і подібні до них;

– оперативна пам’ять – не менше 512 Mb;

– середовище функціонування – операційна система сімейство Windows;

– вимоги до техніки безпеки при роботі з програмою повинні відповідати існуючим вимогам та стандартам з техніки безпеки при користуванні комп’ютерною технікою.

6. Вимоги до програмної документації

6.1 Обов’язкова поетапна інструкція для майбутніх користувачів, наведена у пункті 3.4

7. Вимоги до технічного захисту інформації

7.1 Необхідно забезпечити захист розроблюваного програмного засобу від несанкціонованого використання.

7.2 Неможливість отримання доступу незареєстрованих користувачів до інформаційних ресурсів.

8. Техніко-економічні показники

8.1 Цінність результатів використання даного проекту повинна перевищувати витрати на його реалізацію.

8.2 Має бути реалізований таким чином, щоб підходити для використання широкого загалу.

9. Стадії та етапи розробки

№	Назва та зміст етапу	Термін виконання	
		початок	закінчення
1	Визначення напрямку бакалаврської роботи, формулювання теми	20.03.2025	23.03.2025
2	Аналіз предметної області обраної теми	24.03.2025	13.04.2025
3	Розробка алгоритму роботи	14.04.2025	27.04.2025
4	Написання бакалаврської роботи на основі розробленої теми	28.04.2025	25.05.2025
5	Передзахист бакалаврської кваліфікаційної роботи	26.05.2025	27.05.2025
6	Виправлення, уточнення, корегування бакалаврської дипломної роботи	28.05.2025	02.06.2025
7	Захист бакалаврської кваліфікаційної роботи	09.06.2025	10.06.2025

10. Порядок контролю та прийому

10.1 До приймання бакалаврської кваліфікаційної роботи надається:

– ПЗ до бакалаврської кваліфікаційної роботи;

- демонстрація результату бакалаврської кваліфікаційної роботи;
- презентація;
- відзив керівника роботи;
- відзив рецензента

Технічне завдання до виконання прийняв _____ Порицька В.В

Додаток Б. Лістинг коду розробленого додатку

```
const express = require('express');
const dotenv = require('dotenv');
const cors = require('cors');
const cookieParser = require('cookie-parser');
const { mongoose } = require('mongoose');
const app = express();
app.use(cors({
   origin: 'http://localhost:5173',
   credentials: true
}));
require('dotenv').config();
mongoose.connect(process.env.MONGO_URL)
   .then(() => console.log('database connected successfully!'))
   .catch(() => console.log('error during connected to database'))
//middle
app.use(express.json());
app.use(cookieParser());
app.use(express.urlencoded({extended: false}))
app.use('/', require('./routes/authRoutes'));
app.use('/product', require('./routes/productRoutes'));
router.post('/register', registerUser);
router.post('/login', loginUser);
router.get('/profile', getProfile);
const registerUser = async (req, res) => {
   console.log('Request body:', req.body);
   try {
      const { name, email, password, role } = req.body;
      if (!name) {
         return res.json({
            error: 'name is required'
         })
      }
      if (!password || password.length < 8 || !/[a-z]/.test(password) ||
         !/[A-Z]/.test(password) ||
         !/[!@#$%^&*(),.?":{}|<>]/.test(password)
      ) {
         return res.json({
           error: 'Password must be at least 8 characters long and include uppercase, lowercase
letters and a special character.'
         });
      }
      const exist = await User.findOne({ email });
      if (exist) {
         return res.json({
            error: 'email is taken alredy'
         })
      }
```

```
            const hashedPassword = await hashPassword(password)

            const user = await User.create({
              name, email, password: hashedPassword, role: "User"
            })
            return res.json(user)
          } catch (error) {
            return res.json({
              error: error
            })
          }
        }
        const loginUser = async (req, res) => {
          try {
            const { email, password } = req.body

            const user = await User.findOne({ email });
            if (!user) {
              return res.json({
                error: 'User dosen`t exist'
              })
            }
            const match = await comparePassword(password, user.password)
            if (match) {
                  jwt.sign({ email: user.email, id: user._id, name: user.name, role: user.role },
process.env.JWT_SECRET, {}, (err, token) => {
                res.cookie('token', token).json(user)
              })
            }
            if (!match) {
              res.json({
                error: 'Password do not match'
              })
            }
          } catch (error) {
            console.log(error)
          }
        }
        const getProfile = (req, res) => {
          const { token } = req.cookies;
          if (token) {
            jwt.verify(token, process.env.JWT_SECRET, {}, (err, user) => {
              if (err) return res.status(401).json({ error: 'Invalid token' });
              res.json(user);
            });
          } else {
            res.json(null)}}}}
        router.post('/add', addProduct);
        router.get('/all', getProducts);
        router.post('./card', getCard);
        router.post('/order', Order);
```

```
router.post('/remove', deleteFromBase);
router.get('/history', getHistory)
router.put('/update/:id', UpdateProduct);
router.post("/confirm", DSecure)
const addProduct = async (req, res) => {
  try {
    const { name, decript, img, price, cuantity, category } = req.body;
    if (!name || !decript || !img || !price || !cuantity || !category) {
      return res.json({
        error: 'all filds is required'}
    }
    const exist = await Product.findOne({ name });
    if (exist) {
      return res.json({
        error: 'Name is shoud be unique'
      })
    }
    const product = await Product.create({
      name, decript, img, price, cuantity, category
    })
    return res.json(product)
  } catch (error) {
    return res.json({
      error: error)ï
const getProducts = async (req, res) => {
  try {
    const products = await Product.find();
    res.json(products);
  } catch (error) {
    res.status(500).json({ error: 'Failed to fetch products' });
  }
};
const getHistory = async (req, res) => {
  try {
    const history = await Transaction.find()
      .populate('product')
      .populate('userId');
    res.json(history);
  } catch (error) {
res.status(500).json({ error: 'Failed to fetch history' });
  }
}
const Order = async (req, res) => {
  try {
    const { card, user, stripeToken } = req.body;
    const userId = user._id || user.id;
    const totalAmount = card.reduce((acc, item) => acc + item.price, 0);
    const lastAttempts = await Transaction.find({
      userId: userId,
      createdAt: { $gte: new Date(Date.now() - 5 * 60 * 1000) }
    })
```

```
            if (lastAttempts.length >= 3) {
              return res.status(429).json({ error: "Too many attempts already tried. Please try again
later." });
            }
    const dailyTransactions = await Transaction.find({
              userId: userId,
              status: "succeeded",
              createdAt: { $gte: new Date(Date.now() - 24 * 60 * 60 * 1000) }
            });
           const dailyTotal = dailyTransactions.reduce((sum, tx) => sum + tx.price, 0);
            const DAILY_LIMIT = 1500;
            if (dailyTotal + totalAmount > DAILY_LIMIT) {
              return res.status(403).json({
                error: `The daily limit of ${DAILY_LIMIT} has been exceeded. Today, we have already
spent ${dailyTotal}`
              });
            }
    const productIds = await Product.find({
              name: { $in: card.map(item => item.name) }
            }).select("_id")
            const highRiskCountries = ['MY', 'MX', 'IN', 'TH'];
            const paymentMethod = await Stripe.paymentMethods.retrieve(stripeToken);
            const cardCountry = paymentMethod.card.country;
            const requireThreeDSecure = highRiskCountries.includes(cardCountry);
            const paymentIntent = await Stripe.paymentIntents.create({
              amount: totalAmount * 100,
              currency: 'usd',
              payment_method: stripeToken,
              confirm: true,
              return_url: 'http://localhost:5173/card',
              payment_method_options: {
                card: {
                  request_three_d_secure: requireThreeDSecure ? 'any' : 'automatic'
                }
              },
              automatic_payment_methods: {
                enabled: true,
                allow_redirects: 'never'
              }
            });
            // Створення первинної транзакції
            const attempt = await Transaction.create({
              name_user: user.name,
              userId: userId,
              price: totalAmount,
              product: productIds.map(product => product._id),
              status: paymentIntent.status
            });
    if (paymentIntent.status === "requires_action") {
              return res.json({
                requires_action: true,
```

```
            client_secret: paymentIntent.client_secret
const updatedTransaction = await Transaction.findByIdAndUpdate(
            attempt._id,
            {
              status: paymentIntent.status,
              product: productIds.map(product => product._id)
            },
            { new: true }
          ).populate('product').populate('userId');

          //  Якщо платіж успішний — зменшуємо кількість товару
          if (paymentIntent.status === "succeeded") {
            await Promise.all(
              card.map(item =>
                Product.findByIdAndUpdate(item._id, {
                  $inc: { cuantity: -1 }}}
          return res.json({
            success: true,
            paymentIntent,
            transaction: updatedTransaction
          });
        } catch (error) {
          console.error('Payment error:', error);
          res.status(500).json({ error: error.message })
    const DSecure = async (req, res) => {
        try {
          const { paymentIntentId } = req.body;
          const paymentIntent = await Stripe.paymentIntents.retrieve(paymentIntentId);
          if (paymentIntent.status === "succeeded") {
            await Transaction.findOneAndUpdate(
              { stripeId: paymentIntent.id },
              { status: "succeeded" }
            );
            return res.json({ success: true });
          }
          return res.status(400).json({ error: "Payment not completed" });
        } catch (err) {
          console.error("Confirm error:", err);
          return res.status(500).json({ error: err.message });}}
    const deleteFromBase = async (req, res) => {
        try {
          const deleteProduct = await Product.findByIdAndDelete(req.body.e)
          console.log(req.body)
          return res.json(deleteProduct)
        } catch (error) {
          console.log(error)     }
    const UpdateProduct = async (req, res) => {
        const { id } = req.params;
        const updatedData = req.body
        try {
```

```
            const updatedProduct = await Product.findByIdAndUpdate(id, updatedData, { new: true
});
            if (!updatedProduct) {
               return res.status(404).json({ error: 'Product not found' })
            res.json(updatedProduct);
         } catch (error) {
            console.error(error);
            res.status(500).json({ error: 'Failed to update product' });
      <Elements stripe={stripePromise}>
         <UserContextProvider>
          <ProductContextProvider>
           <Navbar />
           <div>
            <Toaster position='top-center' toastOptions={{ duration: 3000 }} />
            <Routes>
             <Route path='/' element={<Home />} />
             <Route path='/register' element={<Register />} />
             <Route path='/login' element={<Login />} />
             <Route path='/dashboard' element={<Dashboard />} />
             <Route path='/product/add' element={<AddProduct />} />
             <Route path='/product/card' element={<Card />} />
             <Route path='/product/remove' element={<ProductRemove />} />
             <Route path='/product/history' element={<History />} />
             <Route path='/product/update' element={<Update />} />
             <Route path='/product//user-order' element={<UserOrder />} />
            </Routes>
           </div>
          </ProductContextProvider>
         </UserContextProvider >
        </Elements>
      export function ProductContextProvider({ children }) {
         const { user } = useContext(UserContext);
         const [products, setProdacts] = useState([])
         const navigate = useNavigate()
         useEffect(() => {
            const getProducts = async () => {
               try {
                  const { data } = await axios.get('/product/all');
                  console.log(data)
                  setProdacts(data)
   getProducts();
         }, [])
         const [transaction, setTransaction] = useState([]);
         useEffect(() => {
            const getHistory = async () => {
               const { data } = await axios.get('/product/history')
               setTransaction(data
            }
            getHistory()
         }, [])
```

```
    const [card, setCard] = useState([]);
    const AddToCard = (product) => {
      const IsAlreadyInCard = card.some(p => p._id === product._id)
      if (!IsAlreadyInCard) {
        setCard(prev => [...prev, product]);
        toast.success("Product added in card")
        totalPrice()
      } else {
        toast.error("Product is alredy in card")
    const removeProduct = (item) => {
      setCard(card.filter(prev => prev._id !== item));
      totalPrice()
      toast.success('product was remove!')
const [total, setTotal] = useState(0)
    useEffect(() => {
      const totalPrice = () => {
        setTotal(card.reduce((sum, item) => sum + item.price, 0));
      }
      totalPrice()
    }, [card])

    const Order = async () => {
      try {
        await axios.post('/product/order', { card, user })
          .then(toast.success("Order sent successfully!"));
      } catch (error) {
        console.error("Error sending order:", error);
        toast.error("Failed to send order");
      const deleteFromBase = async (e) => {
      try {
        await axios.post('/product/remove', { e })
          .then(toast.success('Product was remove'));
        const { data } = await axios.get('/product/all');
        setProdacts(data);
      } catch (error) {
        toast.error("Failed to delete product");
      }
    }
   const UpdateProduct = async (e) => {
      try {
        await axios.post('/product/update', {e})
        console.log(e)
      } catch (error) {
        console.log(error)
        toast.error('failed to update')
      }
        return <ProductContext.Provider value={{ products, setCard, AddToCard, card,
removeProduct, total, Order, deleteFromBase, transaction, UpdateProduct, setProdacts }}>
        {children}
      </ProductContext.Provider>
```

Додаток В. Ілюстративний матеріал

Розробка веб-сервісу для захищених інтернет-платежів з інтеграцією платформи Stripe та налаштуванням механізму захисту 3D Secure

Виконала студентка групи 1КІТС-21б Порицька В.В

Науковий керівник: Салієва О.В. д.ф., доц. каф. МБІС

Актуальність роботи

Актуальність впровадження надійних механізмів захисту онлайн-платежів є одним із пріоритетним напрямків у сфері інформаційної безпеки у зв'язку із поширеністю. Тому було розроблено міжнародні стандарти та протоколи, які покликані вирішити цю проблему. Одним із таких протоколів є 3D Secure, який був розроблений для забезпечення додаткового рівня захисту.

Інтеграція платіжної платформи Stripe та запровадження механізму захисту 3D Secure допоможе підвищити рівень безпеки платежів, знизити ризик шахрайства та забезпечити довіру користувачів до онлайн-платежів.

Мета роботи, новизна та практична цінність

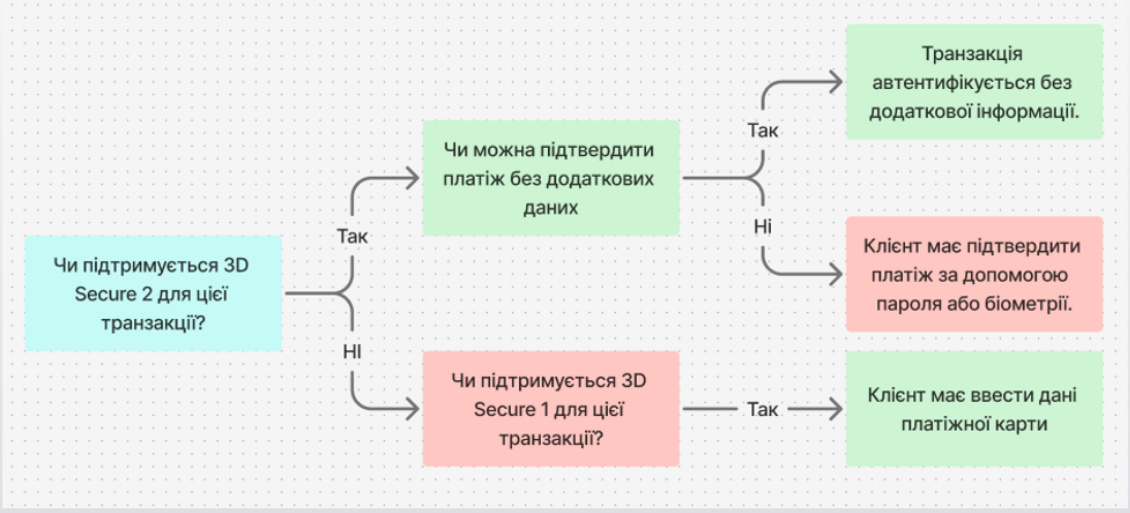
- **Метою роботи** є розробка веб-сервісу для безпечних інтернет-платежів з інтеграцією платіжної платформи Stripe та налаштуванням механізм захисту 3D Secure.
- **Об'єктом дослідження** є процеси здійснення онлайн-платежів та їх захист у веб-сервісах.
- **Предметом дослідження** є методи інтеграції платіжної платформи Stripe та механізму 3D Secure у веб-сервіс для забезпечення безпечних онлайн-платежів.
- **Новизна** полягає у комплексному підході до інтеграції Stripe API та 3D Secure, а також у побудові сценаріїв взаємодії користувача, під час проходження автентифікації.
- **Практична цінність** полягає у використанні розробленого рішення у електронній комерції.

Аналіз існуючих платіжних процесорів

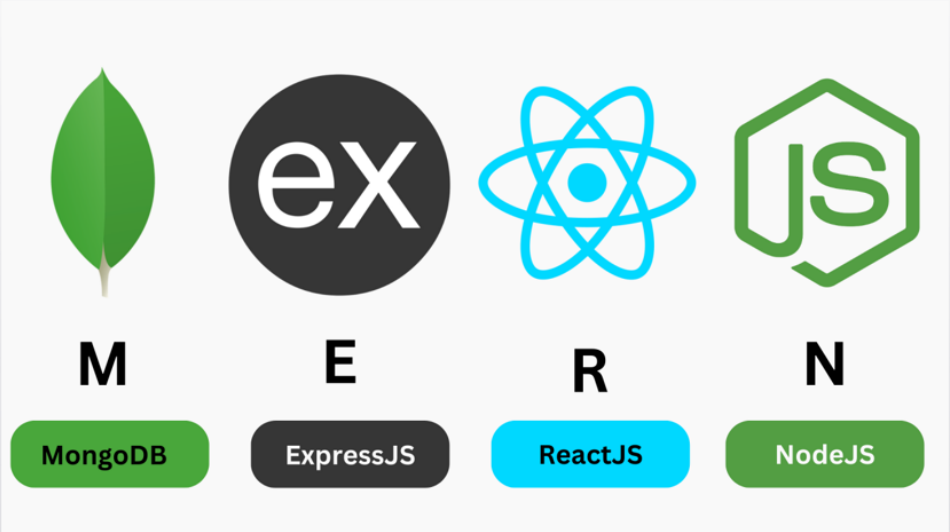
PayPal Braintree 2checkout adyen stripe

Робота 3D Secure

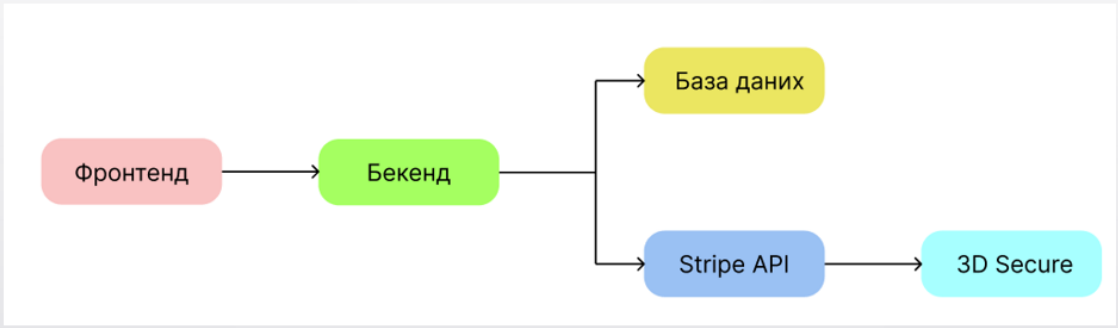
Процес вибору версії протоколу 3D Secure



Використані технології для практичної реалізації сервісу



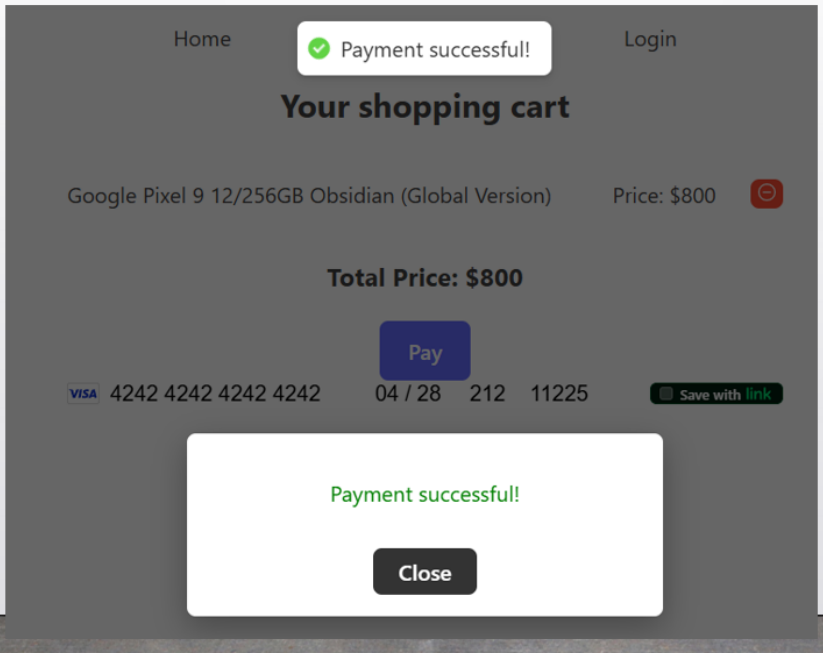
Алгоритм роботи сервісу



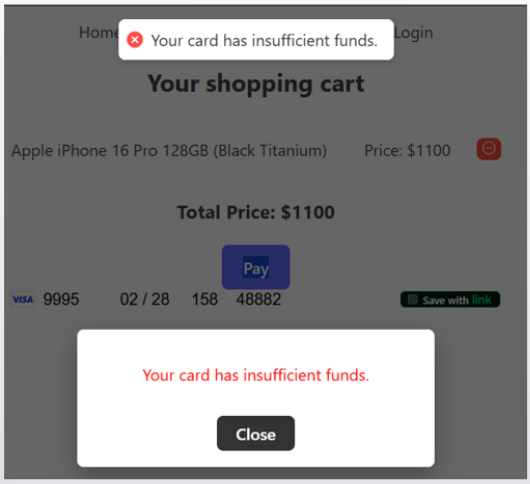
Реєстрація користувачів

Зовнішній вигляд сторінки реєстрації під час взаємодії з користувачем

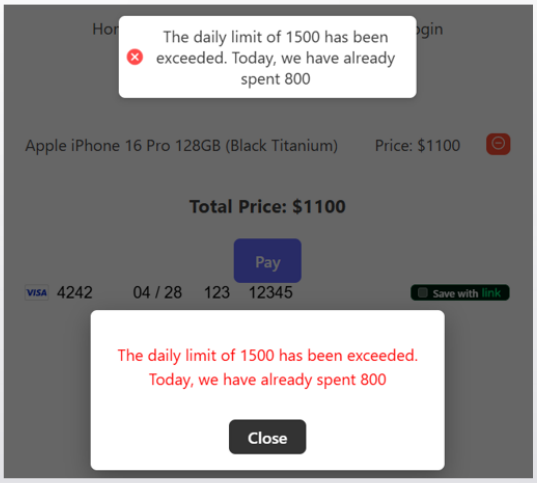
Створення платежу



Перевірка різних сценаріїв

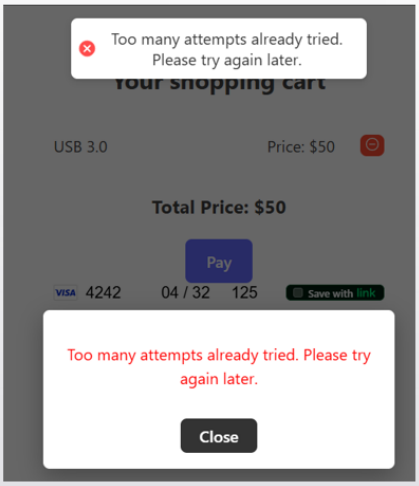


Відхилення платежу через недостатню кількість коштів на рахунку

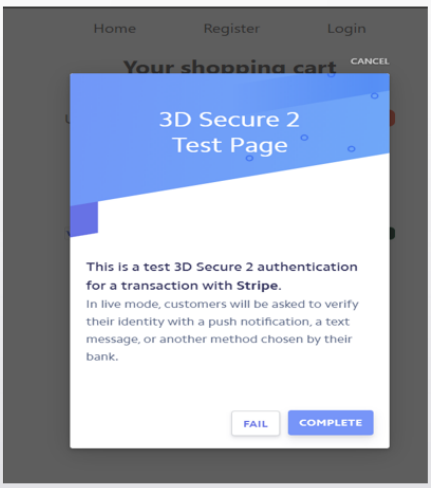


Вигляд вікна при перевищені денного ліміту

Перевірка різних сценаріїв



При четвертій спробі здійснити платіж за короткий проміжок часу



Проходження 3D Secure

Висновки

У ході роботи було:

– проаналізовано сучасні платіжні сервіси, їх переваги та недоліки, особливості забезпечення безпеки у кожному з них;

– досліджено принцип роботи технології 3D Secure, необхідність розвитку різних версій даного стандарту, їх недоліки і переваги. Вплив версії на користувацький досвід та збільшення продажів;

– спроектовано архітектуру веб-сервісу, що включає клієнтську частину – для покращеної взаємодії з користувачем, серверну частину – для безпечної обробки інформації та базу даних – що забезпечує надійне зберігання;

– реалізовано інтеграцію з платформою Stripe, у тому числі із механізмом 3D Secure та налаштуванням додаткових політик безпеки;

– забезпечено збереження статусів платежів у базі даних з урахуванням можливих сценаріїв: успішна оплата, помилка, потреба в додаткових діях, тобто проходження 3D Secure;

– проведено тестування сервісу в тестовому середовищі з використанням тестових карт, яке підтвердило, що поведінка системи стабільна та передбачувана;

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:

Розробка веб-сервісу для захищених інтернет-платежів з інтеграцією платформи Stripe та налаштуванням механізму захисту 3D Secure

Тип роботи: бакалаврська кваліфікаційна робота

Підрозділ Кафедра менеджменту на безпеки інформаційних систем
Факультет менеджменту та інформаційної безпеки
Гр. 1КІТС-21б

Керівник доцент Салієва О.В. _____

Показники звіту подібності

Strike Plagiarism	
Оригінальність	96,98%
Загальна схожість	3,02%

Аналіз звіту подібності (відмітити потрібне)

- ☐ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений (-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ Порицька В.В.
(підпис) (прізвище, ініціали)

Опис прийнятого рішення

- ☐ **Допустити до дахисту**

_____

_____

_____

_____

Особа, відповідальна за перевірку _____ Коваль Н.П.
(підпис) (прізвище, ініціали)

Експерт _____ _____
(за потреби) (підпис) (прізвище, ініціали, посада)