МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА ВЕБІНТЕРФЕЙСУ ДЛЯ СЕРВІСУ МІЖНАРОДНИХ ГРОШОВИХ ПЕРЕКАЗІВ ЗАСОБАМИ VUE JS»**

Виконав: студент 2 курсу, групи 8.1222
спеціальності 122 Комп'ютерні науки
(шифр і назв1а спеціальності)
освітньої програми Комп'ютерні науки
І. С. Решевський
(ініціали та прізвище)

Керівник доцент кафедри комп'ютерних наук, доцент, к.пед.н. Пшенична О.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри програмної інженерії ЗНУ, к.ф.-м.н. Лісняк А.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя
2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти магістр

Спеціальність 122 Комп'ютерні науки
(шифр і назва)

Освітня програма Комп'ютерні науки

ЗАТВЕРДЖУЮ

В.о. завідувача кафедри комп'ютерних наук, д.т.н., професор

_____ Шило Г.М.
(підпис)

« 5 » травня 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Решевському Івану Сергійовичу
(прізвище, ім'я та по-батькові)

1. Тема роботи (проекту) Розробка вебінтерфейсу для сервісу міжнародних грошових переказів засобами Vue JS

керівник роботи (проекту) Пшенична Олена Станіславівна, к.пед.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » травня 2023 року № 642-с

2. Строк подання студентом роботи 29.11.2023

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування та розробка вебінтерфейсу

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 05.05.2023

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи магістра	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	12.05.2023	
2.	Збір вихідних даних.	01.09.2023	
3.	Обробка літературних джерел	5.10.2023	
4.	Розробка першого та другого розділу.	18.10.2023	
5.	Розробка третього розділу.	05.11.2023	
6.	Оформлення та нормоконтроль		
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи магістра.		

Студент _____ І.С. Решевський
(підпис) (ініціали та прізвище)

Керівник роботи _____ О.С. Пшенична
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка вебінтерфейсу для сервісу міжнародних грошових переказів засобами Vue JS»: 68 с., 18 рис., 14 джерел, 4 додатки.

БАНКІВСЬКИЙ РАХУНОК, ВЕБІНТЕРФЕЙС, МІЖНАРОДНІ ГРОШОВІ ПЕРЕКАЗИ, ТОКЕНІЗАЦЯ, ТРАНЗАКЦІЯ, JAVA SCRIPT, REST API, VUE JS

Об’єкт дослідження – системи міжнародних грошових переказів.

Предмет дослідження – вебтехнології для реалізації вебінтерфейсу міжнародних грошових переказів.

Мета роботи: спроєктувати та програмно реалізувати вебінтерфейс системи міжнародних грошових переказів.

Методи дослідження – порівняння, аналіз.

Розроблений вебінтерфейс системи переказу грошей виявила високий рівень надійності та швидкості обробки транзакцій. Ефективність системи була підтверджена серією тестових операцій здійснення переказу грошей при взаємодії серверною частиною проєкту. Проведено успішне впровадження розробленої системи в пілотному режимі в співпраці з обраним партнером.

Розроблений проєкт має широкий спектр застосування в різних галузях, включаючи електронну комерцію, банківську сферу, та мікрофінансові установи. Завдяки своїм універсальним можливостям, система може бути використана в будь-якому бізнесовому середовищі.

SUMMARY

Master’s Qualification Theses « Development of the Web Interface for International Money Transfers Service Using Vue JS»: 68 pages, 18 figures, 14 references, 4 supplements.

BANK ACCOUNT, WEB INTERFACE, INTERNATIONAL MONEY TRANSFER, TOKENIZATION, TRANSACTION, JAVA SCRIPT, REST API, VUE JS

The object of reseach is the system of international money transfers.

The subject of research is web technologies for implementing a web interface for international money transfers.

The aim of research: to design and software implement the web interface of the international money transfer system.

Research methods - comparison, analysis.

The developed web interface of the money transfer system revealed a high level of reliability and transaction processing speed. The effectiveness of the system was confirmed by a series of test operations of money transfer when interacting with the server part of the project. The developed system was successfully implemented in pilot mode in cooperation with the selected partner.

The developed project has a wide range of applications in various industries, including e-commerce, banking, and microfinance institutions. Thanks to its universal capabilities, the system can be used in any business environment.

Specific steps are proposed for the further development of research, improvement of the system and expansion of its functionality.

ЗМІСТ

ВСТУП

Проблема розробки вебсистеми міжнародних грошових переказів є досить актуальною і важливою сьогодні. Вебсистеми міжнародних грошових переказів можуть зробити процес пересилки коштів більш зручним та доступним для користувачів, клієнти зможуть здійснювати перекази в будь-який час та з будь-якого пристрою з доступом до Інтернету.

Вебсистеми можуть дозволити швидше обробляти та здійснювати грошові перекази, скорочуючи час, необхідний для пересилки коштів між різними країнами. Використання вебсистем значно знижує вартість міжнародних грошових переказів, оскільки включає менше посередників. Сучасних технології шифрування та безпеки для забезпечення конфіденційності та захисту грошових операцій роблять перекази коштів значно безпечнішими та надійнішими.

Зростання міжнародної торгівлі, подорожей та взаємодії вимагає ефективних та зручних рішень для грошових переказів між різними країнами. Саме тому багато країн та регіонів працюють над полегшенням регуляторних обмежень для міжнародних грошових переказів через вебсистеми, що створює можливості для таких вебінновацій.

Усі ці фактори підкреслюють важливість розробки вебсистем міжнародних грошових переказів, які відповідають потребам сучасних клієнтів та допомагають забезпечити більше зручність, швидкість та ефективність у глобальних фінансових операціях.

Сучасні системи грошових переказів мають відповідати наступним вимогам:

– системи мають забезпечувати миттєві або максимально швидкі грошові перекази, особливо для термінових виплат та міжнародних операцій;

– вимога до зниження комісій та вартості операцій для користувачів, щоб зробити грошові перекази більш доступними та економічно вигідними;

– інтуїтивний та легкий для використання інтерфейс, який дозволяє користувачам здійснювати операції без складнощів;

– системи повинні бути доступними для користувачів з усього світу та підтримувати різні валюти та мови;

– забезпечення надійної захисту особистої інформації та фінансових даних користувачів за допомогою сучасних технологій шифрування та аутентифікації;

– надання користувачам доступу до історії операцій, статусу переказів та інших деталей для забезпечення прозорості та довіри;

– можливість здійснювати грошові перекази через мобільні додатки та інші мобільні пристрої;

– забезпечення автоматизованих процесів для прискорення обробки переказів та зменшення ризику помилок;

– надання ефективної підтримки клієнтів, включаючи можливість отримання допомоги через чати, телефонну підтримку та електронну пошту.

Ці вимоги враховують потреби користувачів у сучасному світі та відображають тенденції розвитку глобальних фінансових послуг.

В рамках кваліфікаційної роботи було сформульовано наступну мету: спроєктувати та програмно реалізувати вебінтерфейс системи міжнародних грошових переказів засобами мови програмування фреймворку Vue JS.

Об'єктом даної кваліфікаційної роботи є системи міжнародних грошових переказів.

Предметом роботи є вебтехнології для реалізації вебінтерфейсу міжнародних грошових переказів.

Для досягнення поставленої мети сформульовано наступні задачі для їх реалізації:

а) провести аналіз потреб користувачів і вимог до системи міжнародних грошових переказів;

б) розробити архітектуру та дизайн веб-інтерфейсу системи;

в) інтегрувати Vue.js та необхідні залежності;

г) створити компоненти для відображення головної сторінки, форми для введення даних, списку транзакцій, тощо;

д) реалізувати маршрутизацію між компонентами та реалізувати форми для введення даних;

е) налагодити роботу з API на основі моделей даних;

ж) інтегрувати API для додавання платіжних карток у систему;

з) додати валідацію введених даних та повідомлення про помилки;

и) реалізувати обробку помилок, які можуть виникнути під час взаємодії з API;

к) провести тестування функціональності та відображення на різних пристроях;

л) створити документацію для користувачів та розробників, описати функціонал та використані технології.

Структура і обсяг роботи. Магістерська кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел. У першому розділі проведено огляд електронних систем міжнародних грошових переказів. У другому розділі досліджено вимоги до вебінтерфейсу, проєктовано архітектури для взаємодії вебсистеми із сервером та розробка сценарії взаємодії користувачів із системою. Третій розділ присвячено програмній реалізації.Загальний обсяг магістерської роботи – 50 сторінки, 11 рисунків. Список використаних джерел нараховує 13 найменувань.

1 ОГЛЯД ЕЛЕКТРОННИХ СИСТЕМ МІЖНАРОДНИХ ГРОШОВИХ ПЕРЕКАЗІВ

Електронна платіжна міжнародна система [1] – це інфраструктура та сервіси, що дозволяють здійснювати міжнародні фінансові та грошові операції через електронні канали зв'язку, такі як Інтернет, мобільні додатки та інші цифрові платформи. Ці системи створюють можливість для надання фінансових послуг на глобальному рівні, спрощуючи процеси платежів, переказів та управління коштами між різними країнами та валютами.

До основних характеристик електронних платіжних міжнародних систем відносять:

– глобальний досяг. Електронні платіжні системи дозволяють здійснювати операції між різними країнами та валютами, що робить їх ідеальними для міжнародних торгівельних операцій та глобальних фінансових послуг;

– зручність. Користувачі можуть здійснювати операції за допомогою комп'ютера, смартфона чи іншого мобільного пристрою, що робить процес зручним та доступним;

– швидкість. Електронні платіжні системи дозволяють миттєві або дуже швидкі грошові перекази та платежі, особливо у порівнянні зі звичайними банківськими операціями;

– надійність. Ці системи використовують сучасні технології шифрування та захисту, щоб забезпечити безпеку фінансових даних та транзакцій;

– доступність. Більшість електронних платіжних систем надають можливість вибору різних способів оплати, таких як кредитні та дебетові картки, банківські рахунки, електронні гаманці тощо.

На сьогоднішній день існує багато різних систем грошових переказів, які допомагають користувачам здійснювати грошові операції між різними країнами. Розглянемо основні з них.

1.1 Western Union

Western Union [2] (рисунок 1.1) – це одна з найвідоміших та широко використовуваних міжнародних платіжних систем. Вона дозволяє користувачам здійснювати грошові перекази та платежі між різними країнами та валютами. Western Union має велику мережу агентств та пунктів обслуговування по всьому світу, де клієнти можуть надсилати та отримувати гроші. Основною метою Western Union є надання можливості безпечної та зручної передачі грошей через міжнародну мережу.

Рисунок 1.1 – Логотип системи переказів Western Union

Western Union дозволяє переказувати гроші через агентство, онлайн платформу або мобільний додаток.

Час доставки грошових переказів може варіюватися від миттєвого до кількох годин або днів, залежно від обраного методу та місця призначення.

Western Union зазвичай стягує комісії за свої послуги, які можуть відрізнятися в залежності від суми переказу, місця призначення та обраного методу.

Забезпечує безпеку фінансових операцій за допомогою технологій шифрування та аутентифікації.

Western Union часто використовується для передачі грошей між різними країнами, надання фінансової допомоги родичам та друзям, а також для сплати за товари та послуги у різних країнах.

1.2 MoneyGram

MoneyGram [3] (рисунок 1.2) є глобальним провайдером фінансових послуг, який надає можливість відправляти та отримувати гроші через свою мережу пунктів обслуговування.

Рисунок 1.2 – Система грощових переказів MoneyGram

MoneyGram надає клієнтські додатки для мобільних пристроїв (iOS та Android) та веб-інтерфейс для доступу до системи через веб-браузер. Клієнти можуть використовувати ці додатки та веб-інтерфейс для відправлення та отримання грошових переказів, перевірки стану транзакцій, реєстрації нового облікового запису та інших фінансових операцій.

Програмна реалізація включає ефективні механізми автентифікації користувачів для забезпечення безпеки та конфіденційності.

Застосовуються сучасні технології шифрування та заходи безпеки для захисту особистих даних та фінансових транзакцій.

Платіжна система MoneyGram взаємодіє з банками та іншими фінансовими установами для здійснення фінансових операцій. Ця інтеграція відбувається через стандартні API та протоколи для ефективного обміну даними та грошовими переказами.

Бекендова частина платіжної системи відповідає за обробку транзакцій та взаємодію з банками та іншими учасниками фінансового процесу.

Це включає обробку запитів на переказ грошей, здійснення перевірок та підтверджень, відслідковування стану транзакцій та операцій зворотного зв'язку з клієнтами.

MoneyGram використовує різноманітні системи моніторингу та антішахрайства для виявлення та запобігання шахрайству та фінансовим злочинам. Інтегровані системи допомагають вчасно виявляти та реагувати на потенційні загрози безпеці.

MoneyGram може досліджувати можливість інтеграції з блокчейн-технологією для поліпшення безпеки, прозорості та швидкості глобальних грошових переказів.

Програмна реалізація включає модулі аналітики та звітності, які дозволяють аналізувати транзакції, оцінювати ефективність та приймати стратегічні рішення.

1.3 Wise (Transfer Wise)

Wise [4] (рисунок 1.3) – популярна міжнародна фінансова платіжна система, яка спеціалізується на переказі грошей між країнами з мінімальними комісійними та конкурентними обмінними курсами.

Рисунок 1.3 – Міжнародна фінансова платіжна система Wise

Wise використовує власний алгоритм для конвертації грошей, що дозволяє користувачам отримувати реальний обмінний курс без прихованих комісій.

Ця система дозволяє відправити гроші в одній валюті та отримати їх у іншій, максимізуючи вигоду для клієнтів.

Щоб інтегрувати сервіс Wise в інші програми або продукти, можна використовувати їхні API. API дозволяють автоматизувати перекази грошей, отримувати інформацію про обмінні курси та інші фінансові дані.

Wise має мобільні додатки для iOS та Android, а також веб-інтерфейс для зручності користувачів. Користувачі можуть здійснювати грошові перекази та керувати своїми фінансами через ці платформи.

Забезпечення безпеки та конфіденційності особистих даних є важливим аспектом програмної реалізації Wise. Застосовуються сучасні технології шифрування та механізми безпеки для захисту особистої інформації та транзакцій.

1.4 PayPal

PayPal [5] (рисунок 1.4) є однією з найвідоміших та найпопулярніших міжнародних платіжних систем, яка надає можливість електронних платежів та грошових переказів через Інтернет.

Рисунок 1.5 – Платіжна система PayPal

PayPal була заснована у 1998 році.У 2002 році PayPal була придбана компанією eBay, що покращило її популярність та розширило функціонал.

PayPal дозволяє користувачам створювати облікові записи, пов'язані з банківськими рахунками чи кредитними картками. Користувачі можуть здійснювати електронні платежі та перекази грошей в межах системи та за її межами.

PayPal надає API для інтеграції з веб-сайтами та додатками, щоб дозволити користувачам здійснювати оплату та отримувати платежі на своїх платіжних сторінках. Це включає платіжні кнопки, інтеграцію з мобільними додатками та платіжні віджети.

PayPal забезпечує високий рівень безпеки та конфіденційності транзакцій. Використовуються сучасні методи шифрування та аутентифікації для захисту особистих даних та фінансових операцій.

PayPal дозволяє відправляти та отримувати гроші між користувачами з різних країн. Валютні операції можуть бути автоматично конвертовані за поточним обмінним курсом.

PayPal має мобільні додатки для iOS та Android, які дозволяють здійснювати оплату та перевірку балансу через мобільні пристрої. Також є веб-інтерфейс для доступу до облікового запису через веб-браузер.

PayPal бере комісію за деякі типи транзакцій, включаючи міжнародні платежі та конвертацію валют. Вартість комісій залежить від типу та обсягу транзакції.

1.5 Revolut

Revolut [6] (рисунок 1.5) є популярною міжнародною фінансовою платіжною системою, яка надає широкий спектр фінансових послуг, включаючи конвертацію валют, глобальні перекази грошей, власні банківські рахунки та багато іншого. Початково це була мобільна програма для обміну валют, а згодом розширилася до повноцінного фінансового сервісу. Revolut дозволяє користувачам створювати багато гаманців у різних валютах та використовувати їх для оплати, здійснення переказів, конвертації валют та багато іншого.

Користувачі можуть здійснювати оплату за допомогою додатків Revolut, фізичних карток або віртуальних карток.

Revolut надає API для інтеграції з веб-сайтами та додатками, щоб дозволити користувачам здійснювати оплату та отримувати платежі на своїх платіжних сторінках. Це включає платіжні віджети, механізми оплати через QR-коди та інші інструменти.

Revolut забезпечує високий рівень безпеки та конфіденційності транзакцій. Застосовуються сучасні методи шифрування та аутентифікації для захисту особистих даних та фінансових операцій.

Revolut дозволяє користувачам відкривати мультивалютні рахунки, де можна зберігати гроші у різних валютах та здійснювати конвертацію в межах додатку за обмінним курсом Revolut.

1.6 Висновки за розділом 1

Таким чином, провівши дослідження кожної з платіжних систем окремо, можна зібрати результати їх аналізу за такими характеристиками: географічне покриття, комісії, швидкість операцій, зручність використання, підтримка для бізнесу зібрані (таблиці 1.1).

Таблиця 1.1 – Загальна характеристика платіжних систем

Платіжна система	**Географічний покриття**	**Комісії**	**Швидкість операцій**	**Зручність використання**	**Підтримка для бізнесу**
Western Union	Глобальне	Високі	Повільні	Складно	Так
MoneyGram	Глобальне	Високі	Повільні	Зручно	Так
Wise (TransferWise)	Глобальне	Середні	Середні	Дуже зручно	Так
PayPal	Глобальне	Різні	Швидкі	Зручно	Так
Revolut	Обмежене (головним чином Європа)	Зазвичай низькі	Швидкі	Дуже зручно	Так

Аналіз наведених результатів та потреб користувачів ринку дозволяє виявити низку причин розробки власної платіжної системи:

а) власна платіжна система надає повний контроль над всіма аспектами процесу оплати та транзакцій;
б) створення власної платіжної системи може зменшити витрати на комісії, які інші платіжні посередники беруть за обробку транзакцій;
в) існує можливість розробити власні стандарти безпеки та захисту;
г) власна платіжна система може легко розширюватися та адаптуватися під зростання обсягу операцій і нові потреби.

Кожна електронна платіжна система має дві обов’язкові компоненти клієнтську та серверну частину. В рамках даної кваліфікаційної роботи проєктується та програмно реалізується вебінтерфейс взаємодії користувачів із сервісом здійснення закордонних платежів.

2 ПРОЄКТУВАННЯ СХЕМИ ВЗАЄМОДІЇ ІЗ СЕРВІСОМ ПЛАТІЖНОЇ СИСТЕМИ

2.1 Визначення функціональних та нефункціональних вимог до вебінтерфейсу

Функціональні та нефункціональні вимоги до веб-інтерфейсу грають ключову роль у розробці успішних та користувацьких дружніх вебсайтів.

Функціональні вимоги, як правило, складаються з:

- user story;
- use cases;
- wireframes.

User Story – це короткий опис функціональної вимоги до програмного продукту з точки зору користувача. Цей підхід використовується в методології розробки програмного забезпечення Agile, для опису функціональності, яку повинен надати програмний продукт, щоб задовольнити потреби користувача. User Story має просту структуру і включає в себе наступні елементи:

Завдання (As a). Визначення користувацької ролі або категорії осіб, які стосуються цієї функціональності. У проэкты це "Як користувач платіжної системи" та "Як адміністратор веб-сайту."

Я хочу (I want). Опис конкретної функціональності або можливості, яку користувач бажає мати. Наприклад, «Я хочу мати можливість здійснювати платіж ці своє платіжної картки», «Я хочу мати можливість здійснювати платіж з гаманцю в системі», «Я хочу мати можливість здійснювати миттєвий платіж».

Щоб (So that): Визначення користі від цієї функціональності або ціль, яку користувач прагне досягти, використовуючи цю функцію. У проєкті це "Щоб легко здійснювати грошовий переказ"

User Stories використовуються для створення списку завдань (беклогу), які потім можна призначити для розробки в конкретних ітераціях (спринтах). Вони допомагають команді розробників краще розуміти потреби користувачів та створювати програмне забезпечення, яке відповідає цим потребам.

Use Cases (сценарії використання) - це методологічний підхід та інструмент для аналізу та опису функціональності програмного продукту з точки зору користувача або зовнішнього системного агента. Вони використовуються для опису взаємодії користувачів з програмою або системою і для уточнення, як програмне забезпечення повинно реагувати на конкретні дії користувачів чи інших акторів.

Функціональні вимоги User Story для веб-інтерфейсу платіжної системи:

– система повинна надавати форму входу з можливістю введення електронної пошти та пароля;

– система повинна надавати можливість відновлення пароля через електронну пошту;

– головна сторінка повинна відображати поточний баланс користувача;

– інформація про баланс повинна оновлюватися в реальному часі під час проведення операцій;

– система повинна надавати форму для введення суми та реквізитів отримувача;

– користувач повинен мати можливість вибрати зі списку контактів або ввести реквізити вручну;

– система повинна надсилати користувачеві повідомлення електронною поштою або в мобільний додаток після кожної успішної транзакції;

– система повинна надавати сторінку історії транзакцій з фільтрами за датою та типом операцій;

– користувач повинен мати можливість налаштовувати автоматичні сповіщення про досягнення певного рівня балансу;

– система повинна дозволяти користувачам змінювати особисті дані, такі як електронна пошта, пароль тощо;

– система повинна надавати інтерфейс для додавання нових карток, видалення та редагування існуючих.

Основні компоненти Use Cases включають:

– актори (Actors). Це особи, групи або зовнішні системи, які взаємодіють з програмним продуктом. Актори можуть бути користувачами, іншими програмами, апаратними пристроями тощо;

– сценарії (Scenarios). Сценарії використання описують конкретні послідовності дій, які виконуються користувачами або іншими акторами під час взаємодії з програмним продуктом. Кожен сценарій описує конкретну дію та очікуваний результат;

– взаємодія (Interaction). Опис взаємодії між акторами та програмним продуктом в контексті конкретного сценарію. Включає в себе вхідні дані, дії та вихідні результати.

Use Cases розроблюваної системи представлено на рисунку 2.1.

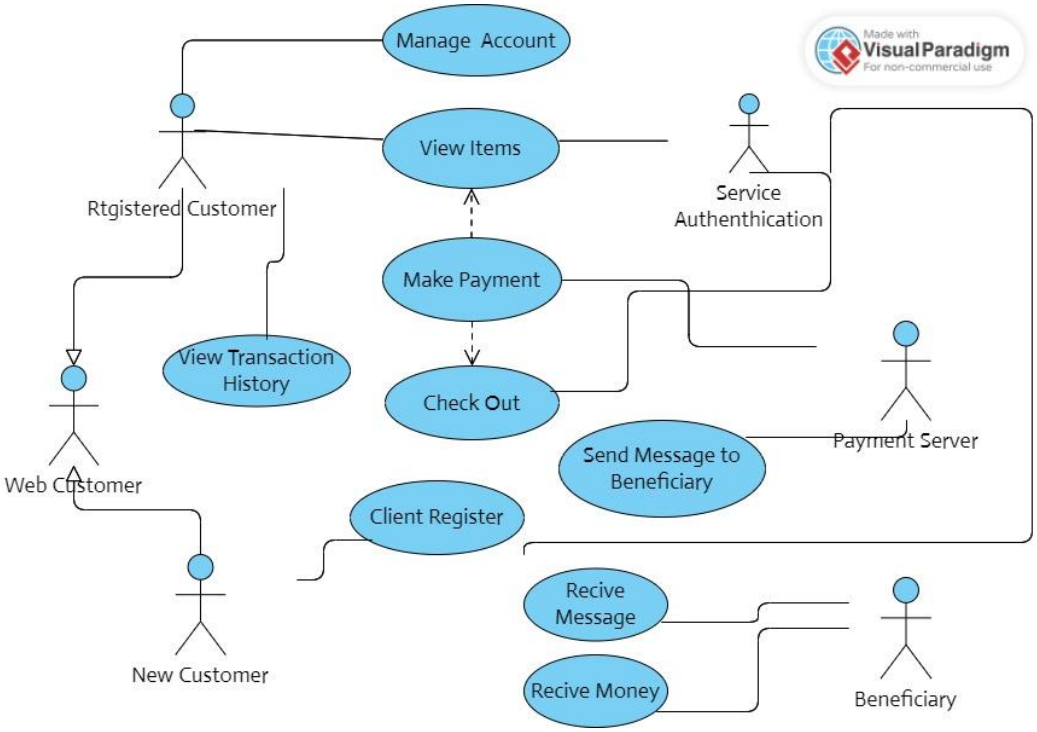

Рисунок 2.1 – Діаграма прецедентів системи

Use Cases допомагають розробникам та аналітикам краще зрозуміти потреби користувачів та визначити функціональність, яка повинна бути реалізована в програмному продукті. Вони також служать основою для створення тестових сценаріїв та перевірки, чи програмне забезпечення відповідає вимогам та очікуванням користувачів.

Wireframes – це структурні та графічні ескізи веб-сторінок або інших елементів інтерфейсу, які використовуються для візуалізації та проектування дизайну веб-сайту, додатка або програмного продукту. Wireframes є однією з перших стадій процесу розробки веб-дизайну і визначають загальну структуру, розташування елементів та інші ключові аспекти інтерфейсу без звернення до деталей дизайну, таких як кольори, шрифти або графічні ефекти.

Основні цілі використання Wireframes включають:

– спрощення комунікації. Wireframes допомагають розробникам, дизайнерам і клієнтам краще розуміти структуру та організацію інтерфейсу. Вони служать інструментом для спільної роботи та обговорення концепцій.;

– визначення логіки та функціональності. Wireframes допомагають визначити розташування та взаємодію елементів, таких як кнопки, поля введення та посилання, а також визначити, як користувачі взаємодіють з інтерфейсом;

– оцінка просторового розміщення. Wireframes відображають структуру сторінок та їхні пропорції, що допомагає визначити оптимальне розміщення елементів;

– тестування концепцій та функціональних можливостей. Wireframes дозволяють перевірити, чи відповідає проектна концепція потребам користувачів та специфікаціям перед початком розробки;

– Wireframes можуть бути намальовані вручну на папері, створені за допомогою спеціалізованих інструментів для розробки веб-дизайну (наприклад, Adobe XD, Sketch, Figma) або навіть створені за допомогою простих графічних програм. Вони допомагають зосередитися на функціональних аспектах дизайну перед виконанням деталей та графічного оформлення.

2.2 Архітектура вебінтерфейсу сервісу грошових переказів

В результаті аналізу вимог до побудови ефективної структури інтерфейсу розроблено проєкт сервісу грошових переказів, який включає ряд компонентів, що забезпечують зручний та ефективний взаємодію користувача з системою:

а) головна сторінка;

б) огляд інформації: короткий огляд основної інформації про обліковий запис користувача, стан рахунку та останні транзакції;

в) форма переказу, яка в ключає форма для введення реквізитів отримувача, суми переказу та іншої необхідної інформації, можливість вибору різних методів переказу, наприклад, банківський переказ, електронні гроші, чек, тощо;

г) перегляд історії транзакцій. Список та деталі попередніх транзакцій, включаючи статус, дату та суму;

д) фільтри та сортування. Можливість фільтрувати та сортувати транзакції для зручності користувача;

е) керування обліковим записом. Можливість зміни особистих даних, таких як адреса, номер телефону чи електронна пошта та паролю;

ж) сповіщення та підтримка. Інформування користувача про статус переказів, зміни в обліковому записі та інші важливі події.

з) служба підтримки. Контактна інформація або чат для звертань до служби підтримки;

и) аутентифікація та авторизація. Засоби для захисту облікового запису та відправлених даних.

Взаємодію наведених компонент представлено не діаграмі рисунку 2.2.

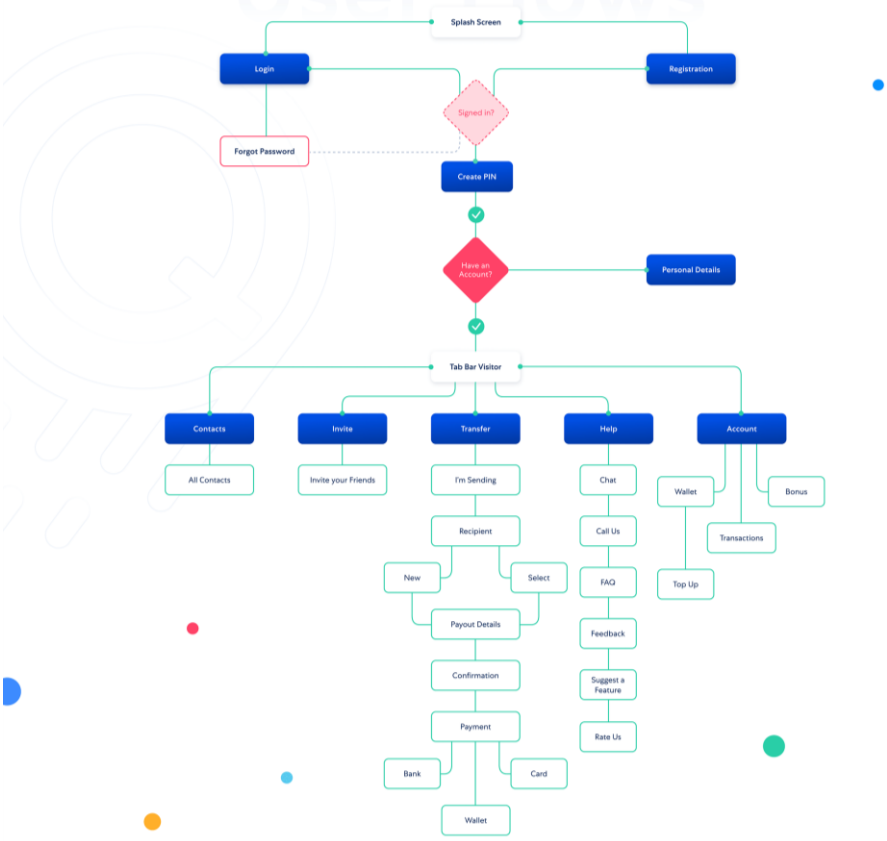

Рисунок 2.2 – Діаграма діяльності користувача на сайті вебсервісу грошових переказів

Розроблена архітектура вебсервісу буде використана при програмній реалізації проєкту.

2.3 Висновки за розділом 2

Етап проектування в розробці програмного забезпечення відіграє ключову роль у формуванні основних аспектів системи та визначенні її архітектури. Він дозволяє ретельно вивчити та визначити всі вимоги до системи перед її реалізацією. Врахування вимог на етапі проектування допомагає уникнути невірного спрямування в подальшому розробці. Розробка архітектури системи на етапі проектування дозволяє визначити структурні компоненти та взаємозв'язки між ними.

Проектування дозволяє ідентифікувати можливі проблеми або складнощі на ранніх етапах розробки, коли виправлення ще є досить простим та витрати на це меншими.

Взаємодія з замовником на етапі проектування дозволяє враховувати їхні вимоги та забезпечувати високий рівень задоволення від кінцевого продукту.

3 РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

3.1 Аналіз технологій та інструментів для програмної реалізації вебінтерфейсу

Розробка вебінтерфейсу динамічних та інтерактивних вебдодатків відбувається з використанням JavaScript-фреймворків. З кожним роком ринок JavaScript-фреймворків розширюється, пропонуючи різноманітні інструменти та технології для реалізації задач.

Розглянемо кілька з найпопулярніших фреймворків для фронтенд-розробки на мові JavaScript: React, Angular, Vue.js та Svelte – чотири з найбільш визнаних та використовуваних фреймворків у світі фронтенд-розробки. Кожен з цих інструментів має свої унікальні особливості, які можуть впливати на ефективність розробки та якість кінцевого продукту.

Докладно розглянувши їх можливості, спільноту, екосистему та нюанси використання, зробимо інформований вибір фреймворку для реалізації поставленої задачі.

3.1.1 Фреймворк React

React [7] – це бібліотека для розробки інтерфейсів користувача (UI) вебдодатків, яка була розроблена Facebook і виведена на ринок у 2013 році. Однією з ключових особливостей React є використання віртуального DOM для ефективного оновлення інтерфейсу, що сприяє підвищенню продуктивності. Бібліотека ґрунтується на компонентній архітектурі, де інтерфейс користувача розділяється на невеликі та самостійні компоненти, полегшуючи розробку та утримання коду.

React використовує JSX (JavaScript XML) для декларативного опису інтерфейсу прямо в коді JavaScript, що полегшує розуміння та модульність коду. Його спрощений підхід дозволяє розробникам легко створювати високоефективні односторінкові додатки (SPA), де сторінка завантажується один раз, а зміни відбуваються без повторного завантаження.

Однією з сильних сторін React є активна та велика спільнота розробників. Велика кількість бібліотек та інструментів, які розроблені спільнотою, роблять React привабливим вибором для розробки. Разом із цим, React має розширюваний інтерфейс, що дозволяє легко інтегрувати його з іншими технологіями та інструментами.

Загальною рисою React є його декларативний підхід, швидке оновлення за допомогою віртуального DOM та фокус на компонентній архітектурі. Ці функції роблять React потужним інструментом для розробки сучасних веб-додатків.

React використовує компоненти як основний будівельний блок для створення інтерфейсу. Це дозволяє розбити великі додатки на менші частини, які легко керуються. Використовуючи віртуальний DOM, React зменшує кількість операцій оновлення, що призводить до покращеної продуктивності та швидкості рендерингу.

Завдяки однозв'язному потоку даних, React полегшує управління станом компонентів та передачу даних між ними.

Легко розуміти та управляти завдяки компонентній структурі.

Але для новачка React може бути перевантаженою з великою кількістю виборів та можливостей. У великих додатках управління станом може стати більш складним завданням.Також фреймворк вимагає часу для освоєння для розробників, що не знайомі з реактивним підходом та JSX.

Отже, React залишається одним з лідерів у світі фронтенд-розробки завдяки своїм прогресивним підходам та великій спільноті. Він ідеально підходить для створення масштабованих, швидких та динамічних веб-додатків.

3.1.2 Фреймворк Angular

Angular [9] – це високопродуктивний та потужний фреймворк для розробки веб-додатків, що базується на мові програмування TypeScript. Заснований на концепції модульності та компонентної архітектури, Angular надає розробникам ефективний інструментарій для створення складних та масштабованих клієнтських додатків.

Цей фреймворк визначається великою готовністю до використання, пропонуючи розширений набір інструментів для створення SPA (односторінкових додатків) та динамічних інтерфейсів. Angular використовує компіляцію шаблонів під час розробки, що призводить до високої продуктивності та оптимізованої швидкодії.

Однією з ключових особливостей є реактивний підхід до програмування, який сприяє автоматичному оновленню інтерфейсу при зміні стану даних. Інші важливі аспекти фреймворку включають вбудовану систему залежностей, що полегшує управління складністю коду, і механізми для роботи з HTTP-запитами, формами та маршрутизацією.

Angular дозволяє розробникам писати код за допомогою TypeScript, що сприяє роботі над великими проектами, завдяки статичній типізації та високому рівню автоматизації. Інтеграція з різними інструментами розробки і тестування також робить Angular популярним вибором для великих корпоративних проектів.

Основними особливостями фреймворку є:

а) використання TypeScript для статичної типізації та інструментів для підтримки об'єктно-орієнтованого програмування;

б) повний набір інструментів для розробки великих та складних додатків;

в) компонентна архітектура та розширений CLI (Command Line Interface);

г) велика спільнота та підтримка від Google;

д) використовується у великих корпоративних проектах.

Отже, Angular вирізняється потужними можливостями, великою спільнотою розробників і підтримкою від Google, що робить його ефективним інструментом для створення високоякісних та масштабованих веб-додатків.

3.1.3 Vue.js

Vue.js [11] – це прогресивний JavaScript-фреймворк для розробки інтерфейсів, який визначається легкістю використання та простотою інтеграції. Заснований на концепції відділення моделі, виду та контролера (MVVM), Vue.js дозволяє створювати ефективні та динамічні односторінкові додатки (SPA) та інтерфейси для веб-сайтів.

Один із головних плюсів Vue.js полягає в його простоті вивчення, що робить його ідеальним вибором для початківців та тих, хто шукає легкий шлях у світ веб-розробки. Фреймворк дозволяє поетапне впровадження в проект, і ви можете використовувати стільки або наскільки мало Vue.js, як вам необхідно в конкретному випадку.

Vue.js вражає декларативним підходом до програмування і використання синтаксису шаблонів, що легко читається та розуміється. Фреймворк також надає потужні можливості для реактивного програмування, дозволяючи ефективно відслідковувати та оновлювати стан додатку при зміні даних.

Vue.js вражає екосистемою розширень та плагінів, що спрощує інтеграцію з іншими інструментами та бібліотеками. Це також дозволяє розробникам індивідуалізувати свій досвід роботи з фреймворком, використовуючи лише ті функції та модулі, які їм необхідні.

Призначений для створення легких та ефективних інтерфейсів, Vue.js не нав'язує строгих конвенцій і дозволяє вам вибирати, як ви бажаєте організовувати свій код. Завдяки цим рисам, Vue.js став популярним вибором для невеликих та середніх проектів, а також для розробників, які шукають гнучкість та простоту веб-розробки.

Особливостями використання фреймворку є:

а) легка вивченість та інтеграція в існуючі проекти;

б) прогресивний підхід дозволяє використовувати лише необхідні функції;

в) розширюється за допомогою плагінів;

г) зростаюча спільнота та підтримка;

д) зручна документація.

3.1.4 Svelte

Svelte [13] – це інноваційний фреймворк для розробки веб-додатків, який відзначається унікальним підходом до створення користувацьких інтерфейсів. Він відрізняється від інших фреймворків тим, що весь його код компілюється в чистий JavaScript в процесі збірки, що призводить до легшого та ефективнішого виконання веб-додатків.

Однією з ключових особливостей Svelte є відсутність використання бібліотеки у результуючому коді. Фреймворк генерує оптимізований, нативний JavaScript-код під час компіляції, що дозволяє зменшити обсяг фронтенд-бандлів та прискорити завантаження додатків. Це робить Svelte привабливим для тих, хто цінує велику продуктивність та ефективне використання ресурсів.

Svelte пропонує декларативний синтаксис, схожий на інші сучасні фреймворки, такі як React та Vue, що полегшує розробку. Однак, в межах цього синтаксису, Svelte використовує концепцію компонентів та директив для організації логіки та взаємодії в додатках.

Svelte також підтримує реактивність, дозволяючи автоматично відслідковувати зміни даних та оновлювати інтерфейс. Це сприяє створенню динамічних та живих веб-додатків з інтуїтивно зрозумілим кодом.

Важливою перевагою є зменшення кількості вихідного коду та більш прозорий процес розробки завдяки компіляції. Завдяки цьому, Svelte може служити ефективним інструментом для розробників, які прагнуть оптимізувати продуктивність та покращити швидкість роботи своїх веб-додатків.

Особливостями використання фреймворку є:

– компіляція до чистого JavaScript під час збірки;

– відсутність потреби у великій кількості коду на стороні клієнта;

– простий синтаксис та низька вартість зборки;

– зростаюча спільнота, але менша порівняно з іншими фреймворками;

– швидке оновлення та вдосконалення.

3.1.5 Обгрунтування вибору інструменту розробки

Для розробки вебінтерфейсу сервісу міжнародних переказів було обрано фреймворк VueJS. Цей вибір грунтується на:

– реактивоісті та швидкодії. Vue.js надає потужні інструменти для реактивного програмування, що особливо корисно для систем обміну грошей, де швидкість та точність обробки транзакцій є критичними. Реактивність Vue.js сприяє автоматичному оновленню інтерфейсу при зміні стану даних, що робить користування системою більш інтуїтивно зрозумілим для користувачів;

– гнучкості та розширюваності. Vue.js дозволяє розробникам використовувати тільки ті компоненти та функції, які необхідні для конкретного проекту. Це робить фреймворк гнучким та легко адаптованим до змін у вимогах проекту чи ринкових умов;

– інтеграції з іншими технологіями. В системі міжнародних грошових переказів важливо мати можливість інтегрувати Vue.js з іншими технологіями, такими як серверні частини, бази даних та служби безпеки. Vue.js легко інтегрується з різними інструментами, що полегшує розробку та супровід системи;

– активній спільноті та підтримці: Vue.js має широку та активну спільноту розробників. Це забезпечує доступність підтримки, додаткових ресурсів, а також дозволяє швидко знаходити відповіді на технічні питання;

– інтернаціоналізації та міжнародній підтримці. Vue.js має вбудовану підтримку для інтернаціоналізації, що сприяє створенню інтерфейсу, який може бути легко адаптованим для користувачів з різних країн та мовних груп.

Загалом, Vue.js володіє тими характеристиками, які можуть зробити його ефективним інструментом для розробки веб-інтерфейсу системи міжнародних грошових переказів. Vue.js отримав підтримку від багатьох великих компаній та проєктів, що свідчить про його стабільність та довготривалу життєздатність.

3.2 Реалізація вебінтерфейсу для ініціювання та виконання закордонних платежів

Основні скрипти реалізації взаємодії клієнтської частини сервісу з API серверу викладено у додатках А-Г. Розклянемо інтерфейс взаємодії системи із клієнтом.

Першим кроком взаємодії з ситемою є регєстрація для нових киристувачів (див. рис.3.1) або авторизація для вже існуючих клієнтів (див. рис. 3.2).

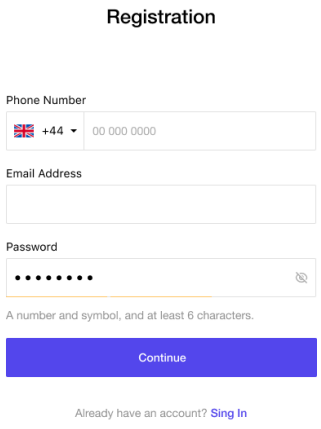

Рисунок 3.1 – Форма реєстрації нових клієнтів

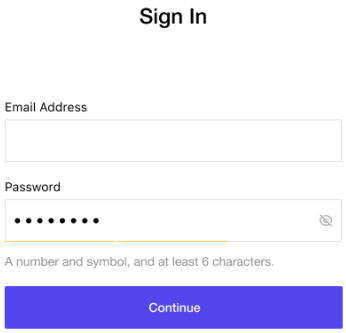

Рисунок 3.2 – Форма авторизації клієнтів

При здійснені переказу (див. рис. 3.3) задаються такі параметри, як сума переказу, країна отримувача (ресіпієнта) та спосіб отриманнія грошей.

Існує три способи отримання грошей:

а) Bank Deposit (Депозит на банківський рахунок). Ресіпієнт може отримати гроші шляхом депозиту на свій банківський рахунок. Важливо вказати реквізити банківського рахунку ресіпієнта, такі як номер рахунку та назва банку. Гроші з'являться на банківському рахунку ресіпієнта після обробки банком;

б) Mobile Transfer (Мобільний переказ). Ресіпієнт може отримати гроші на свій мобільний гаманець або банківський рахунок, пов'язаний з мобільним номером. Зазвичай, ресіпієнт отримає повідомлення про транзакцію та може використати мобільний додаток чи код для підтвердження отримання грошей;

в) Cash Collection (Отримання готівки). Отримання грошей особисто у відділенні платіжної системи або у партнерському пункті обслуговування. Для отримання готівки ресіпієнт може пред'явити ідентифікаційні документи та отримати суму, яку йому було відправлено.

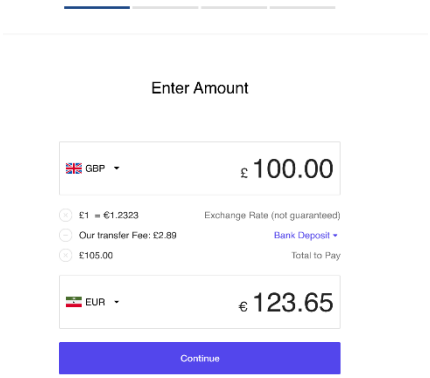

Рисунок 3.3 – Форма вводу початкових параметрів платежу

Наступним кроком є або вибір ресіпієнта зі списку вже існуючих отримувачів (див. рис.3.4), яким раніше здійнювались перекази або створення нового (див.рис.3.5).

При додаванні нового отримувача необхідно ввести наступні дані:

– ПІБ рісіпієнта;

– номер телефону;

– назву організації;

– номер рахунку.

Цей перелік параметрів різниться для різних видів здійснення грошового перказу.

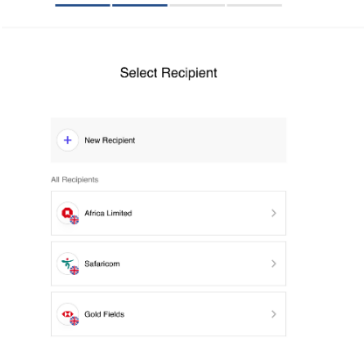

Рисунок 3.4 – Форма вибору ресіпієнта

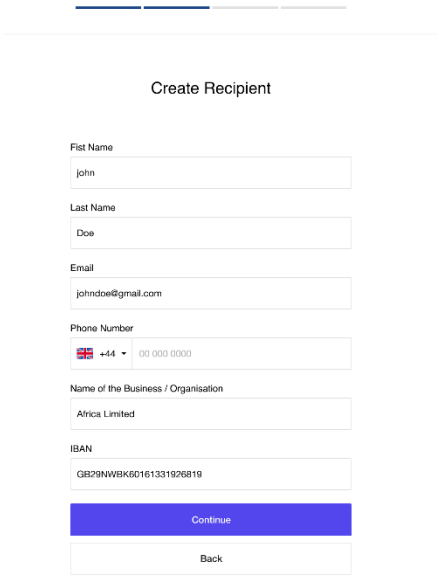

Рисунок 3.5 – Форма введення даних ресіпієнта

Далі введені дані перглядаються та у разі некоректного їх введення, корегуються (див.рис. 3.6).

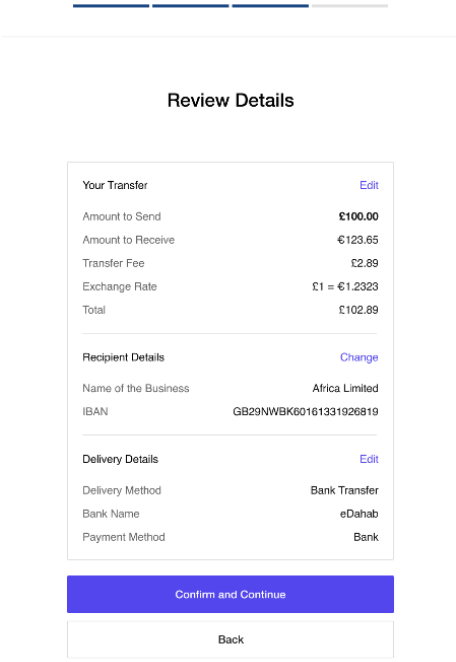

Рисунок 3.6 – Фінальна форма введення параметрів переказу

На фінальній формі транзакції клієнт здіснює оплату переказу одниі із способів:

– миттєвим банківським платежем;

– через гаманець у системі;

– з кредитної картки.

При використанні миттєвого банківського платіжу та кредитної картки використовуються сторонні сервіси.

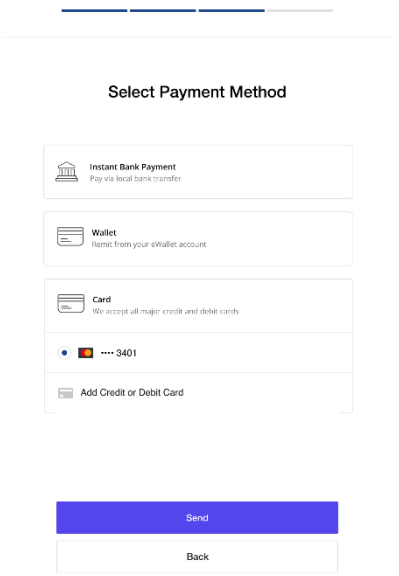

Рисунок 3.7 – Фінальна форма транзакції

При обранні способу платіжу через кредитну картку клієнт вводить данні банківської картки у модальному вінкні на риснку 3.8.

У випадку завершення транзакції сервіс виконує запит у API серверу для перевірки успішності її завершення, після позитивної відповіді користувачеві з’являється повідомлення, представлене на рисунку 3.9, при негативному статусі транзакції з’являється форма на рисунку 3.10. Цей статус перевіряється впродовж 1 хвилини, у випадку успішності оплати користувач отримує форму рисунку 3.9.

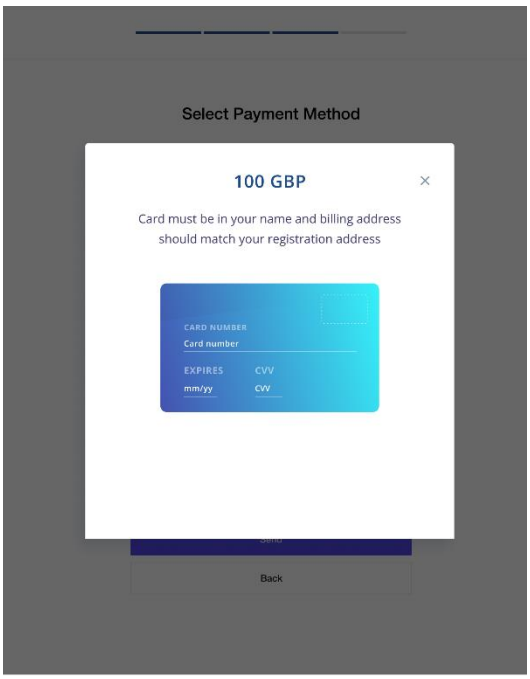

Рисунок 3.8 – Форма введення даних банківськиої картки

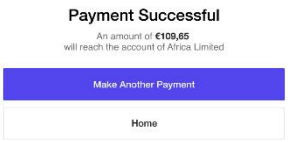

Рисунок 3.9 – Форма успішного завершення транзакції

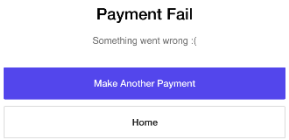

Рисунок 3.10 – Форма неуспішного завершення транзакції

Отже, інтерфейс здійснення переказу інтуїтивно зрозумілий та відповідає стандартам безпечності здійснення фінансових операцій.

3.3 Інтерфейси для ведення та відображення фінансової інформації

Фінансова інформація клієнта відображається у його кабінеті (див. рис. 3.10, 3.11). Вона містить:

– загальну суму грошей на рахунку гаманця

– історію транзакцій;

– інформацію про отримувачів;

– статус транзакцій.

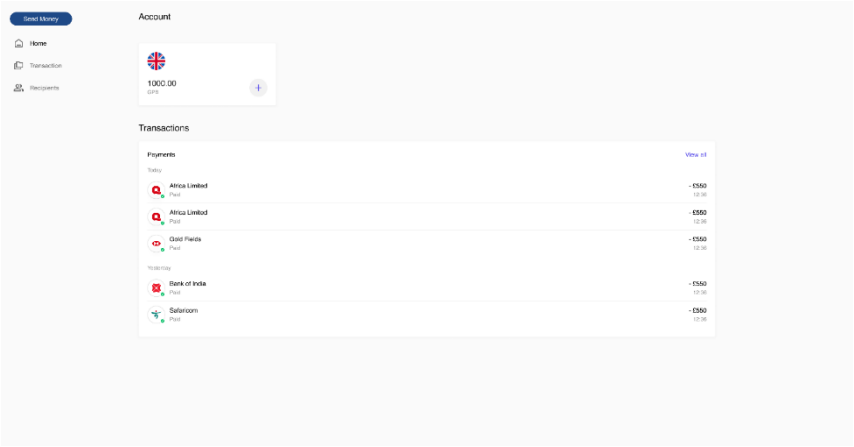

Рисунок 3.10 – Електронний кабінет клієнта сервісу

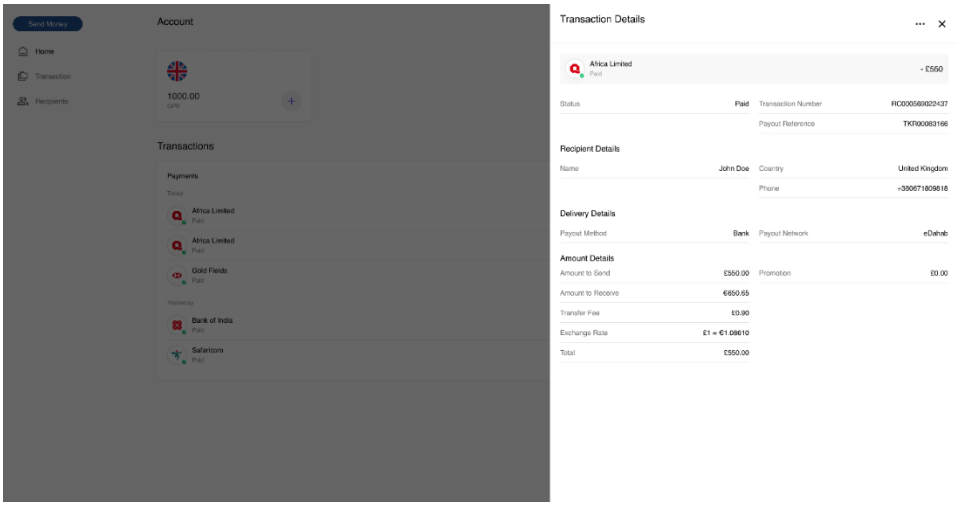

Рисунок 3.11 – Відомості про здійснені транзакції

Інтерфейс відображення фіннасової інформації також інтуїтивно зрозумілий та забезпечує зручність користувачів при аналізі даних і результатів переказів.

3.4 Висновки за розділом 3

Розроблений вебінтерфейс сервісу міжнародних переказів грошей демонструє високий рівень ефективності та швидкості обробки транзакцій, забезпечуючи користувачам миттєвий доступ до своїх фінансових операцій.

Інтерфейс відзначається інтуїтивним дизайном, що сприяє легкому користуванню навіть для новачків. Чітко визначені елементи керування та логічно побудована навігація дозволяють користувачам швидко зорієнтуватися в системі.

Вебінтерфейс успішно адаптований для різних платформ та пристроїв, забезпечуючи користувачам зручний доступ до сервісу як на комп'ютерах, так і на мобільних пристроях.

Високий рівень функціональності дозволяє користувачам не лише переказувати гроші, але й отримувати детальну інформацію про свої фінансові операції. Розроблені розширені можливості сприяють зручності управління фінансами.

Ці висновки свідчать про успішну реалізацію програмного вебінтерфейсу сервісу міжнародних переказів грошей, враховуючи важливі аспекти зручності, функціональності та технологічної передовості.

ВИСНОВКИ

У кваліфікаційній роботі було спроєктувати та програмно реалізувано вебінтерфейс системи міжнародних грошових переказів засобами мови програмування фреймворку Vue JS. Для досягнення поставленої мети вирішено наступні задачі:

а) проведено аналіз потреб користувачів і вимог до системи міжнародних грошових переказів;

б) розроблено архітектуру та дизайн вебінтерфейсу системи;

в) інтегровано Vue.js та необхідні залежності;

г) створено компоненти для відображення головної сторінки, форми для введення даних, списку транзакцій, тощо;

д) реалізовано маршрутизацію між компонентами та реалізовано форми для введення даних;

е) налагоджено роботу з API на основі моделей даних;

ж) інтегровано API для додавання платіжних карток у систему;

з) додано валідацію введених даних та повідомлення про помилки;

и) реалізовано обробку помилок, які можуть виникнути під час взаємодії з API;

к) проведено тестування функціональності та відображення на різних пристроях;

л) створено документацію для користувачів, описано функціонал та використані технології.

ПЕРЕЛІК ПОСИЛАНЬ

1. Електронна система грошових переказів. URL : https://uk.wikipedia.org/wiki/%D0%95%D0%BB%D0%B5%D0%BA%D1%82%D1%80%D0%BE%D0%BD%D0%BD%D0%B8%D0%B9_%D0%BF%D0%BB%D0%B0%D1%82%D1%96%D0%B6 (дата звернення: 5.09.2023)

2. *WesternUnion* URL : https://www.westernunion.com/ua/uk/receive.money.html (дата звернення : 4.10.2023)

3. MoneyGram URL : https://moneygram.ua/ (дата звернення : 4.10.2023)

4. TransferWise(Wise) URL : https://wise.com/ua/ (дата звернення : 4.10.2023)

5. PayPal URL : https:// www.paypal.com/ (дата звернення : 4.10.2023)

6. Revolut URL : https://www.revolut.com/ (дата звернення : 4.10.2023)

7. Walker V. React JS: From Basics to Advanced - A Comprehensive 3-in-1 Guide to Effortless Web Development for Beginners, Intermediates, and Experts. Kindle Edition. 2023. 391p.

8. Spadotto S. React Router Ready: Learn React Router with React and TypeScript. Kindle Edition. 2023. 148 p.

9. Freeman A. Pro Angular: Build Powerful and Dynamic Web Apps 5th ed. Edition. Apress. 2022. 905 p.

10. Bampakos A., Deeleman P. Learning Angular: A no-nonsense guide to building web applications with Angular 15, 4th Edition 4th ed. Edition. Packt Publishing. 2023. 446 p.

11. Hanchett E., Listwon B.Vue.js in Action First Edition. Manning. 2018. 375 p.

12. Harber C. Vue.js for Jobseekers: A complete guide to learning Vue.js, building projects, and getting hired (English Edition) 1st Edition, BPB Publications. 2023. 682 p.

13. Irvine D. Svelte with Test-Driven Development: Advance your skills and write effective automate.d tests with Vitest, Playwright, and Cucumber.js. Packt Publishing. 2023. 250 p.

14. Libby A. Practical Svelte: Create Performant Applications with the Svelte Component Framework 1st ed. Edition. Apress. 2021. 340 p.

ДОДАТОК А

Код скрипту додавання картки клієнта

```
<template>
  <div class="container">
    <div class="box">
      <div class="box__header">
        {{ $t('subtitle') }}
      </div>
      <FormErrors :list="state.apiErrors" />
      <SkeletonCardPayment v-if="state.inProgress" />
      <FormAddCard
        v-show="!state.inProgress"
        :send-button-text="$t(sendButtonTextKey)"
        :is-show-cvv="state.isShowCVV"
        :payout-processor="state.payoutProcessor"
        :type="ECardTypes[state.type]"
        :source-country-code="state.sourceCountryCode"
        :remitter="state.remitter"
        :trans-ref="state.transRef"
        @on-card-tokenized="sentCardTokenToApp"
        @on-card-tokenized-failed="sendErrorToApp"
        @on-set-error-messages="setErrorMessages"
      />
    </div>
  </div>
</template>

<script setup>
import FormAddCard from "@/components/forms/FormAddCard/FormAddCard.vue";
import { ECardTypes } from "@/components/forms/FormAddCard/constants";
```

```
import { computed, onMounted, reactive } from "vue";
import { findGetParameter } from "@/utils/helpers";
import Api from "@/utils/serverApi";
import FormErrors from "@/components/forms/FormErrors.vue";
import i18n from "@/plugins/i18n";
import {useI18n} from "vue-i18n";
import SkeletonCardPayment from "@/components/Skeleton/SkeletonCardPayment.vue";
import { provide } from "vue"
const { t } = useI18n();

const state = reactive({
  type: findGetParameter("type") || "wallet",
  application: findGetParameter("app") || "android",
  lang: findGetParameter("lang") || "EN",
  sourceCountryCode: findGetParameter("source_country_code") || "GB",
  token: findGetParameter("token") || "",
  transRef: findGetParameter("trans-ref") || "",
  remitter: null,
  payoutProcessor: "",
  isShowCVV: true,
  inProgress: true,
  apiErrors: [],
});

const sendButtonTextKey = computed(() => {
  return ECardTypes[state.type] === ECardTypes.wallet
    ? "top_up"
    : "send_button";
});

provide('setInProgressStatus', setInProgressStatus);

onMounted(async () => {
  window.ga("create", import.meta.env.VITE_GA_KEY, "auto");
```

```
  i18n.global.locale.value = state.lang.toUpperCase();
  if (state.token) {
    await getPayoutProcessor();
  }
});

function setInProgressStatus(status) {
  state.inProgress = status;
}

function sentCardTokenToApp(card) {
  console.info("sentCardTokenToApp", card);
  try {
    if (state.application === "ios") {
      window.webkit.messageHandlers.processToken.postMessage(card);
    } else {
      app.setSingleUseToken(card);
    }
  } catch (error) {}
}

function sendErrorToApp(message) {
  console.log("sendErrorToApp", message);
  setErrorMessages([message]);
  try {
    if (state.application === "ios") {
      window.webkit.messageHandlers.processTokenError.postMessage({
        message: message,
      });
    } else {
      app.onTokenizationError(message);
    }
  } catch (error) {}
}
```

```
async function getServiceLocator() {
  try {
    const serviceLocatorData = await Api.get(
      import.meta.env.VITE_SERVICE_LOCATOR_TOKEN_URL + state.token
    );
    Api.setAuthBearer(state.token);
    Api.setBaseUrl(serviceLocatorData.data.middleware);
  } catch (error) {
    const errorMessage = error?.response?.data?.message || t("errorText");
    setErrorMessages([errorMessage]);
    throw error;
  }
}
async function getUserData() {
  try {
    const userData = await Api.post("user/get");
    state.remitter = userData.data.data.remitter;
  } catch (error) {
    setErrorMessages([t("errorText")]);
    throw error;
  }
}
async function getPayoutProcessor() {
  if (ECardTypes[state.type] === ECardTypes.beneficiary) {
    state.isShowCVV = false;
    state.payoutProcessor = "checkout";
  } else {
    try {
      await getServiceLocator();

      const payoutProcessorData = await Api.get(
        "remit/options/payout-processor",
        { source_country_code: state.sourceCountryCode }
      );
      state.payoutProcessor = payoutProcessorData.data.data.payment_processor;
```

```
            if (state.payoutProcessor === "checkout") {
              await getUserData();
            }
          } catch (error) {
            const errorMessage = error?.response?.data?.message || t("errorText");
            setErrorMessages([errorMessage]);
            throw error;
          }
        }
      }

      function setErrorMessages(messages){
        state.apiErrors = messages;
      }
</script>
```

ДОДАТОК Б

Код скрипту додавання картки для provider Checkout

```
<template>
 <div>
  <div class="box__body">
   <form id="payment-form-checkout" method="POST" action="javascript:void(0);">
    <div class="card-pay">
     <div class="card-pay__body">
      <div class="row">
       <div class="col-12">
        <div
         class="form-group form-group-card"
         :class="getClassesForField('card-number')"
        >
         <label>{{ $t("card_number") }}</label>
         <div class="position-relative">
          <div
           class="card-number-frame form-control form-control-card"
          />
          <div class="card-icon" :class="[state.iconCardClass]"></div>
         </div>
        </div>
       </div>
       <div class="col-6">
        <div
         class="form-group form-group-card form-group--exp-date"
         :class="getClassesForField('expiry-date')"
        >
         <label>{{ $t("expires") }}</label>
         <div class="expiry-date-frame form-control form-control-card" />
        </div>
```

```
            </div>

            <div v-if="isShowCvv" class="col-6">
              <div
                class="form-group form-group-card form-group--cvv"
                :class="getClassesForField('cvv')"
              >
                <label>CVV</label>
                <div class="cvv-frame form-control form-control-card" />
              </div>
            </div>
          </div>
        </div>
      </form>
    </div>

    <div class="box__footer">
      <button
        class="btn btn--primary"
        :disabled="!state.isValidForm"
        @click="getToken"
      >
        <span>{{ sendButtonText }}</span>
      </button>
    </div>
  </div>
</template>

<script setup>
import { onMounted, reactive, nextTick, inject } from "vue";
import { addScriptToHtml } from "@/utils/helpers";
import { ECardTypes } from "@/components/forms/FormAddCard/constants";
const emit = defineEmits(["on-card-tokenized", "on-card-tokenized-failed"]);
const setInProgressStatus = inject('setInProgressStatus');
```

```
const state = reactive({
  isValidForm: false,
  iconCardClass: "card-icon--empty",
  initTry: 0,
  fields: {
    "card-number": {
      isEmpty: true,
      isFocus: false,
      hasError: false,
    },
    "expiry-date": {
      isEmpty: true,
      isFocus: false,
      hasError: false,
    },
    cvv: {
      isEmpty: true,
      isFocus: false,
      hasError: false,
    },
  },
});

const checkoutConfig = {
  publicKey: publicKey(),
  localization: {
    cardNumberPlaceholder: "",
    expiryMonthPlaceholder: "MM",
    expiryYearPlaceholder: "YY",
    cvvPlaceholder: "",
  },
  style: {
    base: {
      fontFamily: "opensans,Helvetica,Arial,sans-serif",
```

```
      fontSize: "16px",
      color: "#000",
      lineHeight: "24px",
      letterSpacing: "1px",
    },
    placeholder: {
      base: {
        fontFamily: "opensans,Helvetica,Arial,sans-serif",
        fontSize: "16px",
        color: "#000",
        lineHeight: "24px",
        letterSpacing: "1px",
      },
    },
    invalid: {
      color: "#f05692",
    },
  },
  modes: [],
};

const eventCallbackMap = [
  { event: "FRAME_FOCUS", callback: onFocus },
  { event: "FRAME_BLUR", callback: onBlur },
  { event: "FRAME_VALIDATION_CHANGED", callback: onValidationChanged },
  { event: "CARD_VALIDATION_CHANGED", callback: cardValidationChanged },
  { event: "CARD_TOKENIZED", callback: onCardTokenized },
  { event: "CARD_TOKENIZATION_FAILED", callback: onCardTokenizedFailed },
  { event: "PAYMENT_METHOD_CHANGED", callback: onPaymentMethodChanged },
  { event: "READY", callback: onReady },
];

const props = defineProps({
  sendButtonText: String,
  type: String,
```

```
  transRef: String,
  sourceCountryCode: String,
  remitter: Object,
  isShowCvv: Boolean,
});

onMounted(() => {
  initScriptCheckout();
});

function publicKey() {
  const TYPE_KEY = props.type;
  let prefixCountry = 'EU';

  if (['CA', 'GB'].includes(props.sourceCountryCode)) {
    prefixCountry = props.sourceCountryCode;
  }

  const CHECKOUT_KEY = `VITE_CHECKOUT_${TYPE_KEY}_${prefixCountry}_KEY`;
  return import.meta.env[CHECKOUT_KEY];
}

function onFocus(event) {
  state.fields[event.element].isFocus = true;
}

function onBlur(event) {
  state.fields[event.element].isFocus = false;
}

function onValidationChanged(event) {
  let field = state.fields[event.element];
  field.hasError = !event.isValid;
  field.isEmpty = event.isEmpty;
}
```

```
function onReady() {
  setInProgressStatus(false);
}

function cardValidationChanged() {
  state.isValidForm = window.Frames.isCardValid();
}

function onCardTokenized(data) {
  removeAllCheckoutEvents();
  emit("on-card-tokenized", data);
}

function onCardTokenizedFailed() {
  removeAllCheckoutEvents();
  setInProgressStatus(false);
  const message = "Authorization failed";
  emit("on-card-tokenized-failed", message);
}

function onPaymentMethodChanged(event) {
  state.iconCardClass =
    {
      Mastercard: "card-icon--master",
      Visa: "card-icon--visa",
      "American Express": "card-icon--ae",
    }[event.paymentMethod] || "card-icon--empty";
}

function getClassesForField(fieldName) {
  const field = state.fields[fieldName];
  return {
    "no-empty": !field.isEmpty,
    focus: field.isFocus,
    "has-error": field.hasError,
```

```
  };
}

function initCheckout() {
  if (!window.Frames) {
    state.initTry++;
    if (state.initTry <= 5) {
      setTimeout(initCheckout, 500);
    }
    return false;
  }
  const iframes = document.getElementsByName("checkout-frames");
  iframes?.forEach((item) => item.remove());

  if (!props.isShowCvv) {
    checkoutConfig.modes.push(Frames.modes.CVV_HIDDEN);
  }

  window.Frames.init(checkoutConfig);

  addAllCheckoutEvents();
}

function addAllCheckoutEvents() {
  eventCallbackMap.forEach(({ event, callback }) =>
    window.Frames.addEventHandler(window.Frames.Events[event], callback)
  );
}

function removeAllCheckoutEvents() {
  eventCallbackMap.forEach(({ event }) =>
    window.Frames.removeAllEventHandlers(window.Frames.Events[event])
  );
}
```

```
function getToken() {
  if (!state.isValidForm) {
    return false;
  }
  setInProgressStatus(true);

  if (props.type !== ECardTypes.beneficiary) {
    window.Frames.cardholder = {
      billingAddress: {
        addressLine1:
          props.remitter.building_number + " " + props.remitter.street,
        zip: props.remitter.postcode,
      },
    };
  }

  return window.Frames.submitCard().catch((error) => {
    if (error.message === "Card Expired") {
      state.fields.ExpiryDate.hasError = true;
    }
    setInProgressStatus(false);
  });
}

function initScriptCheckout() {
  addScriptToHtml(
    "https://cdn.checkout.com/js/framesv2.min.js",
    "checkout-frame-script",
    async () => {
      await nextTick();
      initCheckout();
    }
  );
}
</script
```

ДОДАТОК В

Код скрипту додавання картки для provider Fiserv

```
<template>
  <div>
    <div class="box__body">
      <form
        id="payment-form-fiserv"
        ref="fiservForm"
        method="POST"
        @submit.prevent="onSubmitFiservForm"
      >
        <div class="card-pay">
          <div class="card-pay__body">
            <div class="row">
              <div class="col-12">
                <div class="form-group form-group-card">
                  <label>{{ $t("card_holder") }}</label>
                  <div
                    id="cc-name"
                    class="card-holder-frame form-control form-control-card"
                    data-cc-name
                  />
                </div>
              </div>

              <div class="col-12">
                <div class="form-group form-group-card">
                  <label>{{ $t("card_number") }}</label>
                  <div class="position-relative">
                    <div
                      id="cc-card"
```

```
            class="card-number-frame form-control form-control-card"
            data-cc-card
           />
           <div class="card-icon" :class="[state.iconCardClass]"></div>
          </div>
         </div>
        </div>

        <div class="col-6">
         <div class="form-group form-group-card form-group--exp-date">
          <label>{{ $t("expires") }}</label>
          <div
           id="cc-exp"
           class="expiry-date-frame form-control form-control-card"
           data-cc-exp
          />
         </div>
        </div>

        <div class="col-6">
         <div class="form-group form-group-card form-group--cvv">
          <label>{{ $t("cvv") }}</label>
          <div
           id="cc-cvv"
           class="cvv-frame form-control form-control-card"
           data-cc-cvv
          />
         </div>
        </div>
       </div>
      </div>
     </div>
    </form>
   </div>
```

```
    <div class="box__footer">
      <button
        class="btn btn--primary"
        @click="getToken"
      >
        {{ sendButtonText }}
      </button>
    </div>
  </div>
</template>

<script setup>
import { onMounted, reactive, inject } from "vue";
import { addScriptToHtml } from "@/utils/helpers";
import Api from "@/utils/serverApi";
import { useI18n } from "vue-i18n";
const { t } = useI18n();
const setInProgressStatus = inject('setInProgressStatus');
const emit = defineEmits(["on-card-tokenized", "on-card-tokenized-failed"]);

const state = reactive({
  iconCardClass: "empty-card",
  fiservPaymentForm: null,
  initFiservTry: 0,
  tempTokens: [],
});

const fiservConfig = {
  fields: {
    card: {
      selector: "[data-cc-card]",
      placeholder: "",
    },
    cvv: {
      selector: "[data-cc-cvv]",
```

```
    placeholder: "",
   },
   exp: {
    selector: "[data-cc-exp]",
    placeholder: "MM/YY",
   },
   name: {
    selector: "[data-cc-name]",
    placeholder: "",
   },
  },
  styles: {
   input: {
    "font-family": "opensans,Helvetica,Arial,sans-serif",
    "font-size": "16px",
    color: "#000",
    "line-height": "24px",
    "letter-spacing": "1px",
   },
   ".card": {},
   ":focus": {},
   ".valid": {},
   ".invalid": {
    color: "#d0d0d0",
   },
   "@media screen and (max-width: 700px)": {
    input: {},
   },
   "input:-webkit-autofill": {},
   "input:focus:-webkit-autofill": {},
   "input.valid:-webkit-autofill": {},
   "input.invalid:-webkit-autofill": {},
   "input::placeholder": {
    color: "#000",
    "font-size": "16px",
```

```
    },
  },
  classes: {
    empty: "empty",
    focus: "focus",
    invalid: "invalid",
    valid: "valid",
  },
};

defineProps({
  sendButtonText: String,
  type: String,
  transRef: String,
  sourceCountryCode: String,
  remitter: Object,
  isShowCvv: Boolean,
});

onMounted(() => {
  initScriptFiserv();
});

function initScriptFiserv() {
  addScriptToHtml(
    import.meta.env.VITE_FISERV_JS,
    "fiserv-paymentjs-script",
    () => {
      initFiserv();
    }
  );
}

function initFiserv() {
  if (!window.firstdata) {
```

```
    if (state.initFiservTry++ <= 5) {
      setTimeout(() => initFiserv(), 500);
    }
    return false;
  }

  const hooks = {
    preFlowHook: (callbackFn) => {
      Api.post("api/fiserv/client-token")
        .then((response) => {
          // values come from authorize-session endpoint
          callbackFn({
            clientToken: response.data.data.client_token,
            publicKeyBase64: response.data.data.public_key,
          });
        })
        .catch((error) => {
          console.dir(error);
          console.log("preFlowHook");
          showFiservErrorMessage();
        });
    },
  };
  setInProgressStatus(true);
  window.firstdata.createPaymentForm({ ...fiservConfig }, hooks, onCreate);
}

function onCreate(paymentForm) {
  state.fiservPaymentForm = paymentForm;
  setInProgressStatus(false);
  paymentForm.on("cardType", onCardType);
}

function onSubmitFiservForm() {
  if (state.fiservPaymentForm) {
```

```
    setInProgressStatus(true);
    state.fiservPaymentForm.onSubmit(onSuccessPaymentForm, onErrorPaymentForm);
  }
}

function onCardType(data) {
  switch (data.brandNiceType || null) {
    case "MasterCard":
    case "Mastercard":
      state.iconCardClass = "card-icon--master";
      break;
    case "Visa":
      state.iconCardClass = "card-icon--visa";
      break;
    case "AmericanExpress":
      state.iconCardClass = "card-icon--ae";
      break;
    default:
      state.iconCardClass = "card-icon--empty";
      break;
  }
}

function onErrorPaymentForm() {
  this.setInProgressStatus(false);
}

function onSuccessPaymentForm(clientToken) {
  if (state.tempTokens.includes(clientToken)) {
    return false;
  }
  this.setInProgressStatus(true);
  state.tempTokens.push(clientToken);

  state.fiservPaymentForm.reset(() => {
```

```
    Api.post("api/fiserv/card-token", { client_token: clientToken })
      .then((response) => {
        const token = response.data.data.card_token;
        emit("on-card-tokenized", { token });
      })
      .catch(() => {
        console.log("onSuccessPaymentForm");
        showFiservErrorMessage();
      })
      .finally(() => ( setInProgressStatus(false) ));
  });
}

function showFiservErrorMessage() {
  emit("on-card-tokenized-failed", t("errorText"));
}

function getToken() {
  onSubmitFiservForm();
}
</script>
```

ДОДАТОК Г

Код скрипту додавання картки для provider Vantiv

```
<template>
  <div>
    <div class="box__body">
      <form id="payment-form-vantiv" method="POST">
        <div
          id="world-pay-frame"
          class="card-pay card-pay--worldpay"
        />
      </form>
    </div>

    <div class="box__footer">
      <button
        class="btn btn--primary"
        @click="getToken"
      >
        {{ sendButtonText }}
      </button>
    </div>
  </div>
</template>

<script setup>
import { inject, onMounted, reactive } from "vue";
import { addScriptToHtml, uniqID } from "@/utils/helpers";
import Api from "@/utils/serverApi";
const setInProgressStatus = inject('setInProgressStatus');
const emit = defineEmits(["on-card-tokenized", "on-card-tokenized-failed", "on-set-error-
messages"]);
```

```
const state = reactive({
  initWorldpayTry: 0,
  eProtectiframeClient: null,
});

const worldPayConfig = {
  paypageId: import.meta.env.VITE_WORLD_PAY_PAYPAGE_ID,
  reportGroup: import.meta.env.VITE_WORLD_PAY_REPORT_GROUP,
  style: import.meta.env.VITE_WORLD_PAY_STYLE,
  timeout: 15000,
  div: "world-pay-frame",
  callback: worldPayCallback,
  showCvv: true,
  months: {
    1: "01",
    2: "02",
    3: "03",
    4: "04",
    5: "05",
    6: "06",
    7: "07",
    8: "08",
    9: "09",
    10: "10",
    11: "11",
    12: "12",
  },
  numYears: 8,
  tabIndex: {
    accountNumber: 1,
    expMonth: 2,
    expYear: 3,
    cvv: 4,
  },
```

```
  placeholderText: {
    cvv: "",
    accountNumber: "",
  },
};

const iframeId = uniqID();

const props = defineProps({
  sendButtonText: String,
  type: String,
  transRef: String,
  sourceCountryCode: String,
  remitter: Object,
  isShowCvv: Boolean,
});

onMounted(() => {
  initScriptWorldPay();
});

function initScriptWorldPay() {
  addScriptToHtml(
    "https://cdnjs.cloudflare.com/ajax/libs/jquery/3.1.0/jquery.min.js",
    "jquery-script",
    () => {
      addScriptToHtml(
        import.meta.env.VITE_WORLD_PAY_initUrl,
        "eProtect-iframe-client3-script",
        () => {
          initWorldPay();
        }
      );
    }
  );
```

```
}

function initWorldPay() {
  if (typeof EprotectIframeClient === "undefined") {
    state.initWorldpayTry++;
    if (state.initWorldpayTry <= 5) {
      setTimeout(() => initWorldPay(), 500);
    }
    return false;
  } else {
    state.eProtectiframeClient = new EprotectIframeClient(worldPayConfig);
    state.eProtectiframeClient.autoAdjustHeight();
    setInProgressStatus(false);
  }
}

function worldPayCallback(response) {
  if (response?.response === "870") {
    let data = {
      paypageRegistrationId: response.paypageRegistrationId,
      expDate: response.expMonth + "" + response.expYear,
      transRef: props.transRef,
      iframeId,
      lastFour: response.lastFour,
      type: response.type,
    };

    Api.post("api/vantiv-transaction/authorization", data)
      .then((response) => {
        setInProgressStatus(false);
        emit("on-card-tokenized", {
          token: response.data.data.cnpToken,
        });
      })
      .catch((errors) => {
```

```
        let message = "Authorization failed";
        if (errors.response && errors.response.data) {
          const errorData = errors.response.data;
          message = errorData.error_message
            ? errorData.error_message
            : errorData.message;
        }
        setInProgressStatus(false);
        emit("on-card-tokenized-failed", [message]);
      });
  } else {
    setInProgressStatus(false);
    emit("on-set-error-messages", [response.message]);
  }
}

function getToken() {
  emit("on-set-error-messages", []);
  setInProgressStatus(true);
  const message = {
    id: iframeId,
    orderId: props.transRef,
  };
  state.eProtectiframeClient.getPaypageRegistrationId(message);
}
</script>
```