МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА

ШВЕЦЬ ХРИСТИНА ІГОРІВНА

Допускається до захисту:
В.о. завідувача кафедри
інформаційних технологій
канд. техн. наук, доцент
_____ Оксана ЗЕЛІНСЬКА
«_____» _____ 2024р.

РОЗРОБКА ФІНАНСОВОГО МОДУЛЮ CRM СИСТЕМИ ДЛЯ КОЛОРИСТІВ

Спеціальність 122 Комп’ютерні науки
Кваліфікаційна (бакалаврська) робота

Керівник:
Римар П. В., старший викладач кафедри
інформаційних технологій

Оцінка:____ / _____ / _____
(бали за шкалою СКТС/за національною шкалою)
Голова ЕК: _____
(підпис)

Вінниця 2024

АНОТАЦІЯ

Швець Х.І Розробка Фінансового модулю CRM системи для колористів. Спеціальність 122 «Комп’ютерні науки», освітня програма «Комп’ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2024.

У кваліфікаційній (бакалаврській) роботі досліджено галузь роботи колористів, їх потреби на основі яких було побудовано фінансовий модуль для CRM системи. Досліджено використання актуальних веб технологій таких як React, TypeScript,Vite, Redux Toolkit.

Ключові слова: Фінансовий Модуль, Колористи, React

ABSTRACT

Shvetc.K.I Development of Financial Module for CRM System for Colorists
Specialty 122 "Computer Science", Educational Program "Computer Science"
Vasyl Stus Donetsk National University, Vinnytsia, 2024.

The qualification (bachelor's) thesis explores the field of work of colorists and their needs, based on which a financial module for the CRM system has been constructed. The use of current web technologies such as React, TypeScript, Vite, and Redux Toolkit has been investigated.

Keywords: Financial Module, Colorists, React

ЗМІСТ

ВСТУП

Актуальність теми дослідження: У сучасному автомобільному світі, де стиль та індивідуальність стають все більшими чинниками вибору, роль колориста, який відповідає за вибір кольорів для покраски автомобілів, стає критичною для задоволення смаків клієнтів та відповідності останнім тенденціям в автомобільній моді. Розробка фінансового модулю у CRM системі для колористів автомобільної індустрії має надзвичайну актуальність у сучасному світі. З ростом популярності автомобільних інновацій та збільшенням попиту на індивідуалізовані підходи до оформлення автомобілів, професійні колористи стають важливими фігурами в автомобільній індустрії.

Колористи не лише відповідають за вибір кольорів для автомобілів, але й виконують ряд інших функцій, таких як підбір матеріалів, розробка дизайну і додаткових опцій для клієнтів. Враховуючи ці обов'язки, належна організація фінансових процесів та обліку є вирішальним аспектом успішної діяльності колористів.

Метою роботи є розробка інноваційного фінансового модулю для CRM системи, спеціально адаптованого до потреб колористів автомобільної галузі та дослідження використання сучасних веб технологій для побудови веб застосунків. Зростання конкуренції та вимог клієнтів до індивідуального підходу до оформлення автомобілів вимагає від професійних колористів не лише вміння підбирати кольори, а й ефективно управляти фінансовими аспектами своєї діяльності. За відсутності спеціалізованого інструменту для ведення фінансового обліку, колористи можуть стикатися з проблемами в управлінні витратами, реєстрації доходів, виписки рахунків та взаєморозрахунків з клієнтами. Розробка цього модулю має на меті не лише оптимізацію робочих процесів колористів, а й забезпечення їхньої ефективної фінансової управлінської діяльності, що відіграє ключову роль у підвищенні конкурентоспроможності та успіху в автомобільній індустрії

Завданням дослідження роботи є:

- Аналіз потреб та особливостей роботи колористів автомобільної галузі: Детальне вивчення характеристик професійної діяльності колористів, їхніх фінансових потреб та вимог до управління клієнтською базою, зокрема, в контексті вибору кольорів для автомобілів.
- Вибір оптимальних технологій та рішень: Визначення технологій та програмного забезпечення, які найбільш відповідають вимогам фінансового модулю для колористів автомобільної галузі.
- Розробка та імплементація функціональності: Створення інтерфейсу та функціоналу модулю, що забезпечать зручність в користуванні та високу продуктивність, зокрема, у контексті підбору кольорів для автомобілів.
- Вивчення актуальних технологій: Дослідження використання новітніх технологій у веб-застосунках для забезпечення оптимальної швидкодії та стабільності.
- Буде проведено аналіз поточного стану фінансового управління у професійній сфері колористів автомобільної галузі, визначено існуючі проблеми та недоліки в обліку фінансів та взаємодії з клієнтами. На основі цього аналізу буде розроблений фінансовий модуль для CRM системи, який враховуватиме специфіку роботи цієї професійної категорії та пропонуватиме інноваційні рішення для їхньої оптимізації, зокрема, у контексті підбору кольорів для автомобілів.

Об’єктом дослідження є фінансовий модуль CRM системи для оптимізації фінансових типових задач колористів.

Предметом дослідження виступає розробка та використання фінансового модулю, дослідження використання новітніх веб технологій у сучасних застосунках.

Основним завданням роботи розробка фінансового модулю є на основі отриманих потреб колористів, який допоможе колористам ефективно здійснювати автоматизацію рутиних фінансових задач, що забезпечить кращу ефективність їх роботи та автоматизує робочі процеси.

Як теоретичне значеннння одержаних результатів будуть теоретичні основи новітніх технологій для написання сучасних та ефективних веб застосунків. Буде проведено аналіз проблем та болей колористів для ефективної побудови фінансового модулю відносно їх потреб та специфіки їх робочих задач.

У результаті дипломної роботи очікується розробка фінасового модулю CRM системи колористів.

РОЗДІЛ 1
АНАЛІЗ ТА ПОТРЕБИ КОЛОРИСТІВ

1.1. Загальний огляд галузі та особливості роботи колористів

Автомобільна індустрія вважається однією з ключових галузей у світі, привносячи інновації, технологічний прогрес та економічний розвиток. Вона охоплює широкий спектр секторів, включаючи виробництво, ремонт та обслуговування автомобілів, а також дизайн та підбір кольорів, які відіграють ключову роль у визначенні зовнішнього вигляду автомобілів.

Колористи в автомобільній індустрії є професіоналами, відповідальними за підбір кольорів та забарвлення кузовів автомобілів. Вони враховують індивідуальні вподобання клієнтів, технічні характеристики автомобілів, а також модні тенденції та культурні впливи. Основні особливості роботи колористів включають:

1. Технічний аспект: Колористи повинні мати глибокі знання про властивості різних типів фарб, їхній вплив на покриття та тривкість.

2. Творчість: Підбір кольорів - це творчий процес, де важливо виявити індивідуальність та стиль клієнта.

3. Комунікація: Ефективна взаємодія з клієнтами та іншими фахівцями в автомобільній галузі.

У роботі колористів існують певні проблеми, з якими вони стикаються:

1. Сезонність: Попит на певні кольори може змінюватися в залежності від сезону та модних тенденцій, що створює виклики в управлінні запасами фарб та матеріалів.

2. Індивідуальність клієнтів: Кожен клієнт має свої унікальні вимоги та смаки, що вимагає від колористів гнучкості та креативності в роботі.

3. Точність кольору: Важливо забезпечити відповідність обраного кольору зразку або вимогам клієнта, що може бути викликом через різні освітлення та матеріали.

4. Велика кількість товару: Колористу слід постійно тримати в голові доволі суттєву кількість інформації щодо клієнтів та їх замовлень, вдалого поєднання кольорів та точної кількості фарб в наявності. Тому наявність певної системи де можливо усе це автоматизувати призвела б до покращення робочих процесів.

5. Обрахунок зарплати та ціни на певну послугу: Колористу постійно потрібно вручну обраховувати доволі суттєву кількість замовлень, кожне замовлення індивідуальне, а ціни на фарби або ж на будь які ще матеріали можуть змінюватися відносно ринкових цін та стану країни, де знаходиться колорист. Усі ці змінні тримати постійно в голові доволі важко, а витрачати час на обрахунок враховуючи усі фактори займатиме доволі велику частину часу, яку б можна було витратити на більшу кількість замовлень або ж клієнтів.

З урахуванням усіх вищеописаних проблем та зростаючих вимог до точності, ефективності та індивідуалізації в автомобільній індустрії, система управління, спеціально призначена для колористів, стає надзвичайно актуальною. Вона допомагає вирішувати проблеми, забезпечує ефективну організацію робочих процесів та підвищує якість послуг, пришвидшує увесь робочий процес в цілому. В нашій системи враховано усі описані вище проблеми колориста та навіть більше.

1.2. Визначення потреб та вимог користувачів до фінансового модулю

Незаперечно, важливо враховувати, що колористу варто спростити його рутинні завдання і розширити їх функціонал для забезпечення більш ефективної роботи. По-перше, до його обов’язків слід включити внесення нових клієнтів до бази даних. Крім того, важливо редагувати та оновлювати інформацію про клієнтів, враховувати їх актуальні контакті дані.

Другий аспект полягає в наданні спеціальних статусів клієнтам залежно від їхнього статусу та індивідуальних потреб. Наприклад, колорист повинен визначити не лише особливих клієнтів, які мають право на постійні знижки, а й тих, хто виявляє інтерес до нових послуг або продуктів компанії. Такий підхід

дозволить підтримувати тісний контакт із клієнтами та забезпечувати їм персоналізований сервіс, що позитивно позначиться на їхній лояльності та задоволеності від обслуговування.

Крім того, для оптимізації робочого процесу колориста важливо розглянути можливість впровадження функціоналу, який дозволить внесення типових змінних, що широко застосовуються в його повсякденній діяльності. Наприклад, включення у систему можливості редагування параметрів, таких як зарплата для його підопічних, урахування ціни за фарбування певних деталей автомобіля та вибір необхідної фарби. Цей функціонал дозволить значно полегшити процес розрахунків і зберегти час колориста, оскільки він зможе швидко й легко вносити необхідні зміни в систему, не вдаючись до повторного введення даних. Крім того, він зможе легко контролювати та коригувати ціни та інші параметри відповідно до змін на ринку або потреб клієнтів. Такий інтегрований функціонал стане важливою складовою для забезпечення ефективності й точності роботи колориста, а також підвищить рівень його професійної компетентності і задоволення від роботи.

Необхідність розрахунку загального та чистого прибутку становить важливу частину функціонування будь-якого бізнесу. Однак, зосередившись на сфері колористів та розвинувши фінансовий модуль, важливо також включити функціонал, який дозволить відображати заробітну плату кожного колориста в залежності від його статусу, коєфіцієнта оплати праці та кількості виконаних процедур, таких як підбір фарби за великими та малими деталями, а також враховувати різноманітні аспекти підбору кольорів та інші фактори.Цей функціонал значно спростить вирішення рутинних завдань управління фінансами, забезпечивши точність та ефективність у визначенні заробітної плати для кожного колориста. Такий підхід дозволить оптимізувати процес розрахунків, забезпечити справедливі умови оплати праці та підвищити мотивацію персоналу до досягнення кращих результатів. Крім того, він дозволить адміністрації бізнесу ефективно відстежувати фінансові показники та приймати обґрунтовані рішення щодо розвитку компанії.

1.3. Аналіз існуючих фінансових інструментів та CRM систем для колористів

У зв'язку з аналізом існуючих CRM систем для колористів на українському ринку виявлено, що наразі відсутні повноцінні рішення, спеціально адаптовані для потреб цільової аудиторії. Хоча деякі загальнопризначені CRM системи можуть бути використані для управління клієнтськими відносинами, вони недостатньо враховують специфічні потреби та процеси, що характерні для професійної діяльності колористів в автомобільній галузі.

Ця ситуація створює простір для введення на ринок України новітньої та спеціалізованої CRM системи, яка відповідає всім вимогам і потребам колористів. Наш продукт, який спрямований на автоматизацію управління клієнтськими відносинами та оптимізацію фінансового обліку для колористів, стає дуже актуальним та важливим у контексті відсутності адекватних альтернатив.

Відмінною рисою нашого продукту є його спеціалізація на унікальних потребах колористів, а також врахування специфіки автомобільної галузі. Ми розробляємо інноваційні рішення, які не лише вирішують існуючі проблеми, а й вперше вводять нові функціональності, спрямовані на підвищення ефективності та якості обслуговування.

Отже, у контексті відсутності повноцінних альтернатив на ринку України, наш продукт займає лідируючу позицію як актуальне та новітнє рішення, спрямоване на вирішення конкретних проблем та відповідаюче вимогам сучасного бізнесу в галузі підбору фарб для автомобілів.

РОЗДІЛ 2
ЗАСТОСУВАННЯ ТЕХНОЛОГІЙ ТА РІШЕНЬ

2.1. Обумовлення вибору технологій для реалізації фінансового модулю

При розробці фінансового модулю важливо зрозуміти, що вибір сучасних технологій відіграє вирішальну роль у досягненні успіху проекту.

Сучасні користувачі веб-додатків очікують від них не лише функціональності, але й сучасного дизайну та високої продуктивності. Вибір застарілих технологій може призвести до того, що наш додаток втратить конкурентоспроможність на ринку через недостатність функціональності або погану продуктивність. Сучасні технології, такі як React, Vue.js, або Angular, надають можливості для швидкої розробки та масштабування додатків. Їх компонентна архітектура, віртуальний DOM та можливості для реактивного програмування дозволяють збільшити продуктивність розробки та підтримувати додаток на високому рівні.

Застосування сучасних фреймворків та бібліотек дозволяє підтримувати високий рівень безпеки та надійності додатку. Багато сучасних технологій включають вбудовані заходи захисту від атак, такі як CSRF або XSS, а також надають можливості для реалізації автентифікації та авторизації користувачів. Використання популярних сучасних технологій забезпечує доступ до широкої спільноти розробників, яка надає підтримку, допомогу та навчальні ресурси. Крім того, активний розвиток та підтримка фреймворків та бібліотек гарантує, що наш проект залишатиметься актуальним та конкурентоспроможним у майбутньому. Вибір застарілих технологій може призвести до труднощів у розвитку та підтримці додатку у майбутньому. Такі технології можуть мати обмежені можливості, бути менш безпечними або важкими у використанні. Крім того, вони можуть втратити підтримку спільноти та розвиток у майбутньому, що призведе до виникнення технічного боргу та необхідності в переписуванні коду для адаптації до сучасних вимог.

Під час розробки модульної частини застосунку, я використала такі технології як React,TypeScript,Vite,Redux Toolkit, Tailwind та Cypress.

React - Обгрунтування вибору React для написання фінансового модулю нашого веб-застосунку полягає в кількох ключових аспектах. По-перше, React є одним з найпопулярніших фреймворків на сучасному ринку веб-розробки і має велику спільноту розробників, що забезпечує підтримку та допомогу. Крім того, React пропонує компонентну архітектуру, що сприяє швидкій розробці та перевикористанню коду.

Використання віртуального DOM в React дозволяє ефективно оновлювати лише ті елементи інтерфейсу, які змінилися, що покращує продуктивність та відгукність додатку. Крім того, React має велику кількість додаткових бібліотек та інструментів, які доповнюють його функціональність та роблять розробку більш ефективною.

Розширюваність та масштабованість React дозволяють легко масштабувати додаток з ростом його складності. Також, React добре інтегрується з іншими сучасними технологіями, такими як TypeScript, GraphQL, або Next.js, що розширює його можливості та дозволяє використовувати потужні можливості цих технологій разом з React.

Загалом, вибір React для написання фінансового модулю обґрунтований через його популярність, ефективність розробки, продуктивність, розширюваність та масштабованість, а також можливість інтеграції з іншими сучасними технологіями.

TypeScript - TypeScript надає можливість типізації коду, що дозволяє виявляти помилки на етапі компіляції, забезпечуючи вищу надійність та підтримку великих проектів. TypeScript також допомагає зробити код більш зрозумілим та документованим за рахунок визначення типів даних, що сприяє покращенню робочого процесу та зменшенню кількості помилок.

У порівнянні з простим JavaScript, TypeScript надає ряд переваг. По-перше, TypeScript дозволяє визначати типи даних, що забезпечує більшу безпеку та надійність коду. В JavaScript типи даних визначаються динамічно, що може

призвести до непередбачуваних помилок під час виконання програми. TypeScript також підтримує передові функціональності, такі як розширені типи даних, інтерфейси, перечислення, що спрощує розробку та підтримку коду.

Крім того, TypeScript дозволяє використовувати автоматичне визначення типів, що полегшує написання коду та підвищує його читабельність. За допомогою TypeScript можна створювати більш структурований та масштабований код, що сприяє покращенню продуктивності розробників та забезпеченню якості програмного забезпечення.

У випадку вибору між TypeScript та простим JavaScript, варто врахувати потреби проекту та вимоги до надійності та розширюваності коду. Хоча JavaScript є більш простим та доступним для початківців, TypeScript може бути більш підходящим варіантом для серйозних проектів, що вимагають великої стабільності та масштабованості. Приклад застосування Typescript продемонстровано на рисунку 2.1

```
const FinancesPageTemplate: React.FC = () => {
  const [isModalOpen, setIsModalOpen] = useState<financeModalType | false>(false);
  const [currentRowProps, setCurrentRowProps] = useState<null | IColorsitDto>(null);
  const [coloristValue, setColoristValue] = useState<null | IEditColoristProps>(null);
  const [isEnableBtn, setIsEnableBtn] = useState<boolean>(false);

  const dispatch = useAppDispatch();
  const openModal = (modalType: financeModalType) => setIsModalOpen(modalType);
  const closeModal = () => setIsModalOpen(false);
  const [saveDiscount, setSaveDiscount] = useState<IDiscountDto[]>([]);
  const [saveFinancial, setSaveFinancial] = useState<IFinancialDto[]>([]);

  const saveHandler = () => {
    if (saveFinancial.length) {
      dispatch(updateFinanciaCategories(saveFinancial))
        .unwrap()
        .then(async () => {
          setSaveFinancial([]);
          await dispatch(getFinanciiaCategories());
          InvoceService.updateInvoceTotalPrice({});
        })
        .catch((e: Error) => toast.error(e.message));
    }

    if (saveDiscount.length) {
      dispatch(updateDiscountCategories(saveDiscount))
        .unwrap()
        .then(async () => {
          setSaveDiscount([]);
          await dispatch(getDiscountCategories());
          await ClientsService.updateClientsDiscount();
          InvoceService.updateInvoceTotalPrice({});
        })
        .catch((e: Error) => toast.error(e.message));
    }
};
```

Рисунок 2.1 – Приклад React компонента фінансових розрахунків з використанням TypeScript

Vite - Обгрунтування вибору Vite для реалізації нашого функціоналу в веб-застосунку полягає в кількох ключових аспектах. По-перше, Vite - це швидкий та мінімалістичний інструмент для розробки веб-додатків, який базується на сучасних технологіях. Використання Vite дозволяє забезпечити високу швидкість розробки завдяки миттєвому ребілду та гарячій перезавантаженню сторінок, що полегшує розробку та тестування додатків.

У порівнянні з іншими інструментами, такими як Webpack або Parcel, Vite має декілька переваг. По-перше, завдяки використанню модульної системи ES Module, Vite може ефективно розподіляти код додатку та зменшувати час завантаження сторінок. Це дозволяє забезпечити кращу продуктивність та швидкість роботи додатку.

Крім того, Vite підтримує використання препроцесорів CSS та JS, таких як Sass або TypeScript, що полегшує розробку та підтримку коду. Завдяки вбудованій підтримці для сучасних фреймворків, таких як React, Vue.js та Svelte, Vite дозволяє легко створювати додатки на будь-якому з цих фреймворків без зайвих зусиль.

Крім того, Vite надає велику кількість розширень та інструментів для підтримки різних аспектів розробки, таких як виробництво, тестування та оптимізація додатків. Його активна спільнота розробників постійно працює над вдосконаленням та підтримкою інструменту, що забезпечує стабільну та надійну роботу додатків.

Отже, вибір Vite для розробки сучасного веб-застосунку є обгрунтованим через його високу продуктивність, швидкість розробки, широкі можливості підтримки різних технологій та активну спільноту розробників. Приклад використання Vite наведено на рисунку 2.2

```
vite.config.ts
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";
import * as path from "path";

// https://vitejs.dev/config/
export default defineConfig({
  plugins: [react()],
  resolve: {
    alias: [
      { find: "@pages", replacement: path.resolve(__dirname, "src/pages") },
      {
        find: "@templates",
        replacement: path.resolve(__dirname, "src/components/templates"),
      },
      { find: "@", replacement: path.resolve(__dirname, "src") },
    ],
  },
});
```

Рисунок 2.2 – Приклад налаштування конфігурації в нашому проекті для колористів

Redux Toolkit – Вибір задля проекту саме Redux технології та в цілому під час побудови сучасних веб застосунків обумовлено декількома аспектами.

По-перше, Redux Toolkit є офіційним інструментом для роботи зі станом додатку в Redux, що забезпечує його надійність та стабільність. Використання Redux Toolkit спрощує управління станом додатку за рахунок використання зручних інструментів, таких як createSlice та createReducer, що дозволяють писати менше коду та зменшити його складність.

У порівнянні з класичним Redux, Redux Toolkit має декілька переваг. По-перше, використання createSlice дозволяє описувати структуру даних та логіку їх оновлення у більш зрозумілому та компактному форматі. Крім того, Redux Toolkit автоматично генерує action creators та reducers для кожного slice, що полегшує розробку та підтримку коду.

Крім того, Redux Toolkit надає можливість використання додаткових інструментів, таких як createAsyncThunk, що дозволяє працювати з асинхронними операціями у Redux, та configureStore, що дозволяє налаштувати стор додатку за власними потребами.

Однією з ключових переваг Redux Toolkit є його підтримка та активний розвиток спільнотою розробників. Інструмент постійно оновлюється та вдосконалюється з урахуванням потреб розробників, що забезпечує його актуальність та сумісність з останніми версіями Redux та інших залежностей.

В фінансовому модулі нашого застосунку активно використовувався Redux саме задля збереження стану компонентів, їх взаємодія та організація загального структури даних.Приклад використання Redux Toolkit наведено на рисунку 2.3

slice.ts

src > redux > finances > slice.ts > ...

```ts
import { createSlice } from '@reduxjs/toolkit';
import { IInitialState } from './common';
import { financesReducer } from './reducer';

const initialState: IInitialState = {
  isLoading: false,
  discountCategories: null,
  financialCategories: null,
};

const { reducer } = createSlice({
  name: 'financial',
  initialState,
  reducers: {},
  extraReducers: financesReducer,
});

export { reducer };
```

Рисунок 2.3 – Приклад ініціалізації Redux Store фінансів

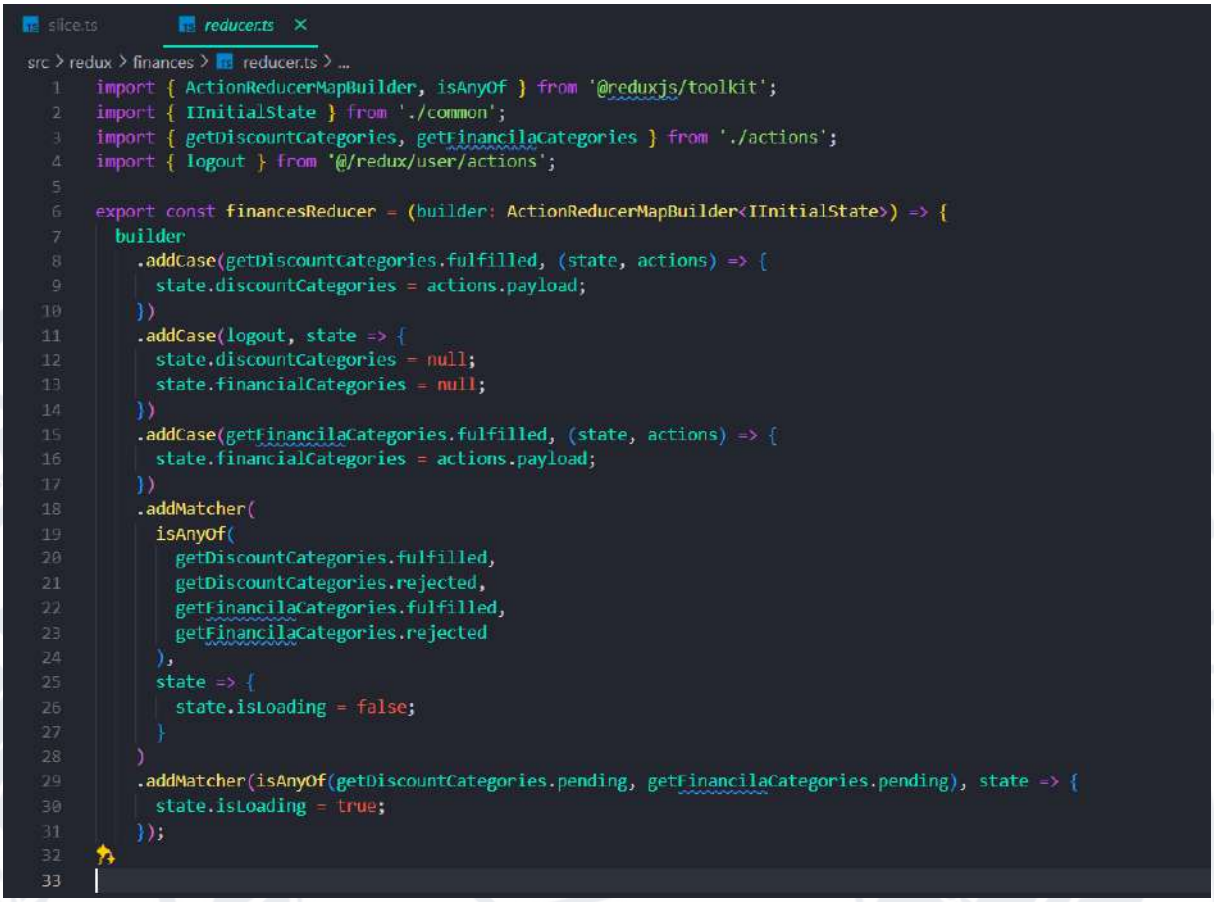


Рис 2.4 – Приклад написання Reducer фінансів для Redux Store

Tailwind – В нашій фінансовій частині застосунку було використано Tailwind CSS задля створення гарного та сучасного інтерфейсу. Ця технологія має декілька ключових причин для вибору під час розробки сучасних застосунків. По-перше, Tailwind CSS пропонує альтернативний підхід до стилізації веб-додатків, що базується на використанні набору готових класів, які описують конкретні властивості CSS. Цей підхід дозволяє швидко та зручно створювати інтерфейс без необхідності власного написання CSS коду.

Однією з переваг Tailwind CSS є його гнучкість та налаштовуваність. Завдяки можливості налаштовувати конфігурацію та розширювати набір класів, Tailwind дозволяє створювати унікальний дизайн, відповідний конкретним потребам проекту. Крім того, використання Tailwind дозволяє підтримувати консистентність дизайну та зменшувати кількість CSS класів у коді, що полегшує його обслуговування та підтримку.

Ще однією важливою перевагою Tailwind є його активна спільнота та екосистема інструментів. Наявність багатофункціональних компонентів, додатків та плагінів розширює можливості веб-розробника та допомагає прискорити процес розробки. Крім того, велика кількість документації та онлайн-ресурсів робить Tailwind доступним для вивчення та використання навіть для початківців.

Отже вибір саме цієї технології є доволі обгрунтованим в нашому модулі. Приклад використання tailwind наведено на рисунку 2.5

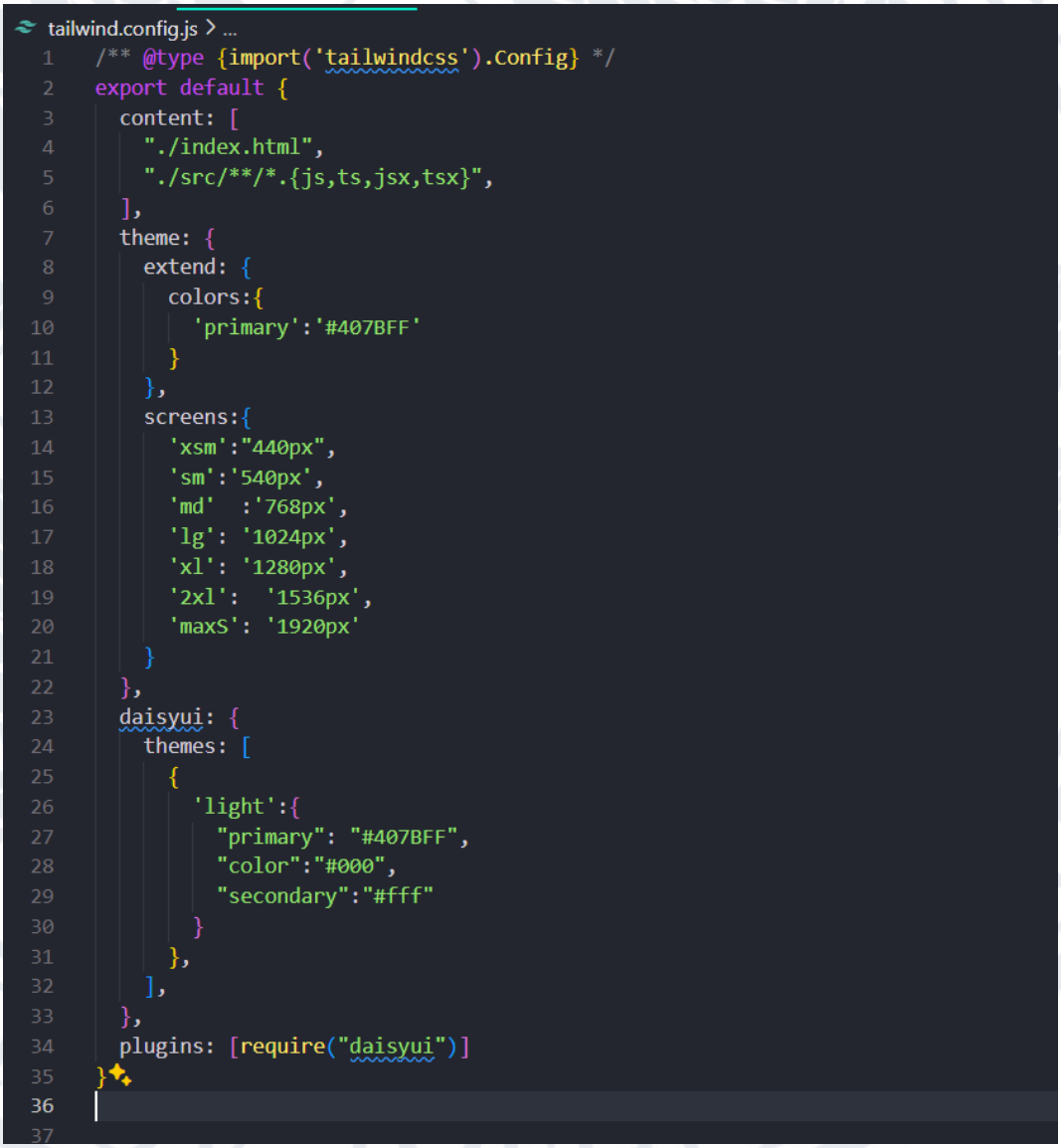


Рис 2.5 – Приклад ініціалізації Tailwind CSS в нашому фінансовому модулі

Cypress – задля написання автотестів в нашому фінансовому модулі я використала також доволі сучасну та актуальну технологію Cypress, яка чудово підходить до нашої мови програмування на проєкті TypeScript.

Обгрунтування вибору Cypress для забезпечення тестування полягає в кількох ключових аспектах. По-перше, Cypress є потужним інструментом для автоматизованого тестування веб-додатків, який забезпечує швидку та ефективну розробку та виконання тестів. Використання Cypress дозволяє забезпечити високу якість та надійність веб-застосунку шляхом автоматизації тестів на різних рівнях, включаючи юніт-тести, інтеграційні тести та тести інтерфейсу користувача.

Однією з ключових переваг Cypress є його зручний та потужний API для написання тестів. Cypress пропонує чистий та експресивний синтаксис, який дозволяє легко описувати тести та взаємодіяти з елементами веб-сторінки. Крім того, Cypress надає багато вбудованих команд для виконання різних дій, таких як клік, заповнення форм, перевірка вмісту тощо, що спрощує написання та підтримку тестів.

Ще однією важливою перевагою Cypress є його здатність до візуалізації тестів та дебагу. Завдяки вбудованому інструменту Cypress Test Runner, розробники можуть переглядати виконання тестів у реальному часі, а також відслідковувати та аналізувати кожен крок виконання тесту. Це полегшує виявлення та виправлення помилок у веб-застосунку та тестах. Приклад використання Cypress наведено на рисунку 2.6

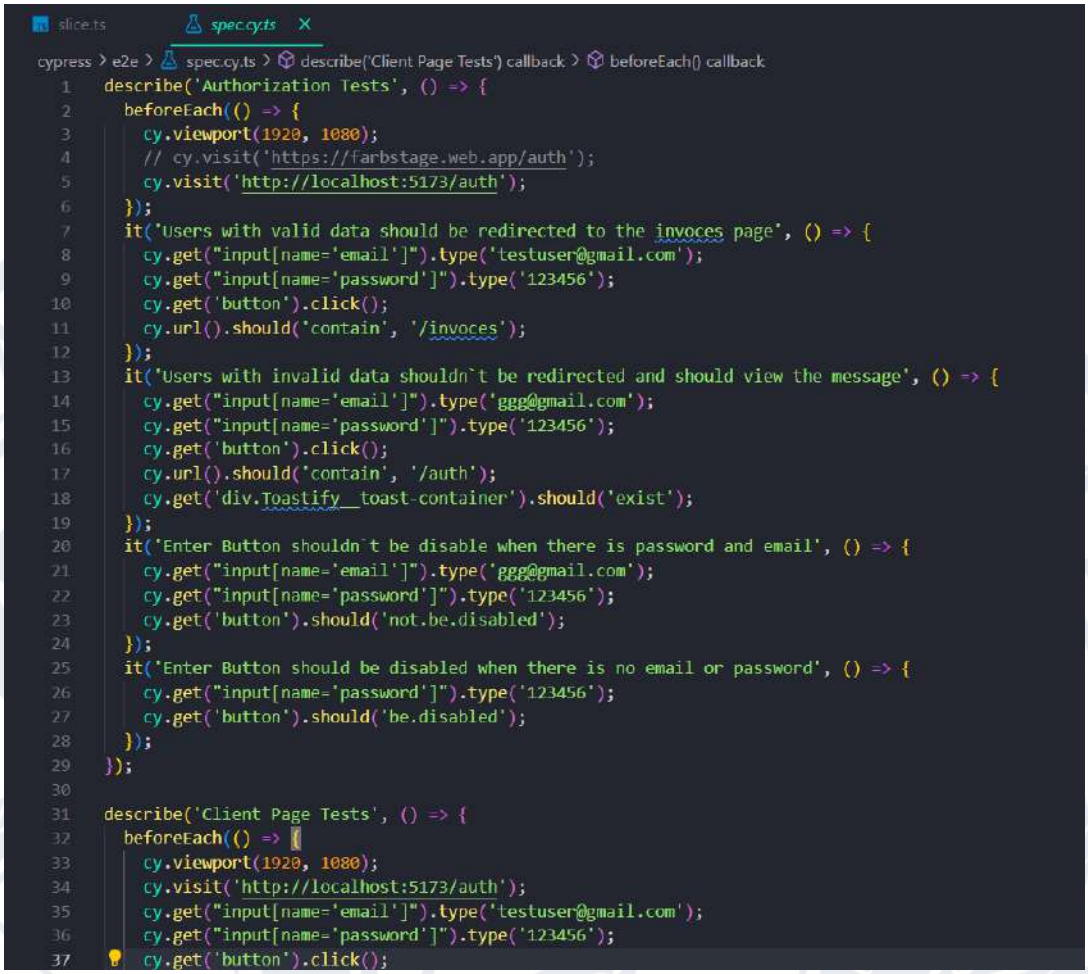


Рисунок 2.6 – Приклад написаних end-to-end тестів для нашого застосунку на Cypress

2.2. Розроблення концепції та функціональності фінансового модулю з урахуванням потреб користувачів

Як вже було описано вище, в нашому фінансовому модулі буде враховано та вирішено наступні актуальні проблеми колористів:

Ведення обліку клієнтів: Модуль забезпечує можливість додавати, редагувати, оновлювати та видаляти дані про клієнтів. Кожному клієнту може бути призначений відповідний статус, що спрощує їхнє управління та класифікацію.

Управління персоналом: Модуль дозволяє додавати нових колористів, встановлювати їм відповідний статус та призначати індивідуальні коди доступу та ставки.

Конфігурація типових змінних: Користувач може налаштувати типові змінні для прорахунку, такі як відсоток знижки для клієнтів з різним статусом (наприклад, VIP, роздрібний або постійний). Також можна встановити вартість різних видів підбору, таких як за габаритними деталями, лючком, автомобілем чи за вибором фарби.

Відображення заробітної плати: Модуль надає зручну можливість відображення підсумків заробітної плати кожного колориста відповідно до встановлених змінних та кількості виконаних операцій за обраний період часу. Взаємодіючи з різними частинами нашого модулю відповідно колористу буде доступні різні функціональні можливості, які ми розглянемо надалі.

2.3 Застосування Flux архітектури під час написання модулю.

2.3.1 Загальне визначення Flux архітектури

Flux - це архітектурний патерн для управління станом додатків, який часто використовується в розробці веб-додатків із використанням бібліотеки React.js. Цей патерн був представлений компанією Facebook як додатковий інструмент для покращення управління станом у великих і складних додатках.

Основна ідея Flux полягає в тому, щоб мати одну централізовану точку керування станом додатку, яку називають "Store". Додаток має взаємодіяти з цим "Store" через деякі визначені дії (actions), які викликаються компонентами. Коли дія викликається, вона передається до "Store", який оновлює свій стан відповідно до отриманих даних.

Один з ключових аспектів Flux - це однонаправлений потік даних. Це означає, що дані рухаються в одному напрямку: від джерела дій, через "Store", до представлення. Це робить архітектуру більш передбачуваною та легкою для розуміння та налагодження.

Ще однією важливою складовою Flux є те, що він покликаний уникнути прямого зв'язку між компонентами. Замість цього, вся комунікація відбувається через дії, які відправляються до "Store", і компоненти споживають дані з "Store", а не одне з одного.

2.3.2 Актуальність та застосування в сучасних веб застосунках Flux архітектури

У сучасних веб-застосунках застосування підходу Flux, особливо в контексті використання бібліотеки React.js, є досить поширеним. Ось кілька причин, чому використання Flux є актуальним і корисним:

- Управління складним станом: Багато сучасних веб-застосунків мають складний стан, який може бути розподілений між багатьма компонентами. Flux надає централізований механізм керування станом, що робить його краще підходящим для складних додатків.
- Однонаправлений потік даних: Однонаправлений потік даних спрощує прогнозованість та розуміння того, як дані рухаються через додаток. Це полегшує відладку і підтримку коду.
- Масштабованість: Архітектура Flux дозволяє легко масштабувати додаток шляхом додавання нових функцій або компонентів без зайвих складнощів. Це робить його популярним в середовищі швидкорозвиваючихся веб-проектів.
- Розподілена команда розробників: Flux робить легше спільну роботу над проектом для розподіленої команди розробників. Чітка структура та однонаправлений потік даних допомагають уникнути конфліктів та забезпечити зручність співпраці.
- Сумісність з React.js: Flux розроблений як компаньйон для React.js, і вони добре доповнюють один одного. React надає відображення інтерфейсу користувача, тоді як Flux забезпечує механізм керування станом додатку.

2.3.3 Використання Redux Toolkit як Flux архітектури на прикладі нашого модулю.

Як було описано вище, Redux Toolkit - це офіційний набір інструментів для роботи з Redux, який спрощує процес використання Redux у ваших додатках та

забезпечує кращу продуктивність та читабельність коду. Redux є однією з реалізацій архітектурного патерну Flux.

Основні принципи Redux, як Flux-подібної архітектури, включають централізоване зберігання стану додатку, однонаправлений потік даних та чітку розділеність між діями, редукторами та станом додатку.

Redux Toolkit надає декілька ключових інструментів для спрощення роботи з Redux:

- configureStore: Це функція, яка допомагає створити Redux store з попередньо налаштованими параметрами, такими як middleware, редуктори та інше.
- createSlice: Цей метод дозволяє визначити "slice" стану та його відповідні дії та редуктори в одному місці, що полегшує роботу зі станом.
- createAsyncThunk та createReducer: Ці утиліти допомагають створити асинхронні дії та редуктори для обробки асинхронних запитів.
- createEntityAdapter: Це інструмент для створення адаптерів сутностей, які полегшують управління нормалізованими даними.

Redux Toolkit робить процес розробки застосунків на основі Redux більш ефективним та продуктивним, зменшуючи кількість рутинних операцій та забезпечуючи кращу організацію коду.

В нашому модулю було застосовано саме цю технологію задля збереження усього стану нашого застосунку та подальшої роботи з ними, редагування, читання та видалення будь яких даних. Приклад такої технології наведено на рисунку 2.7

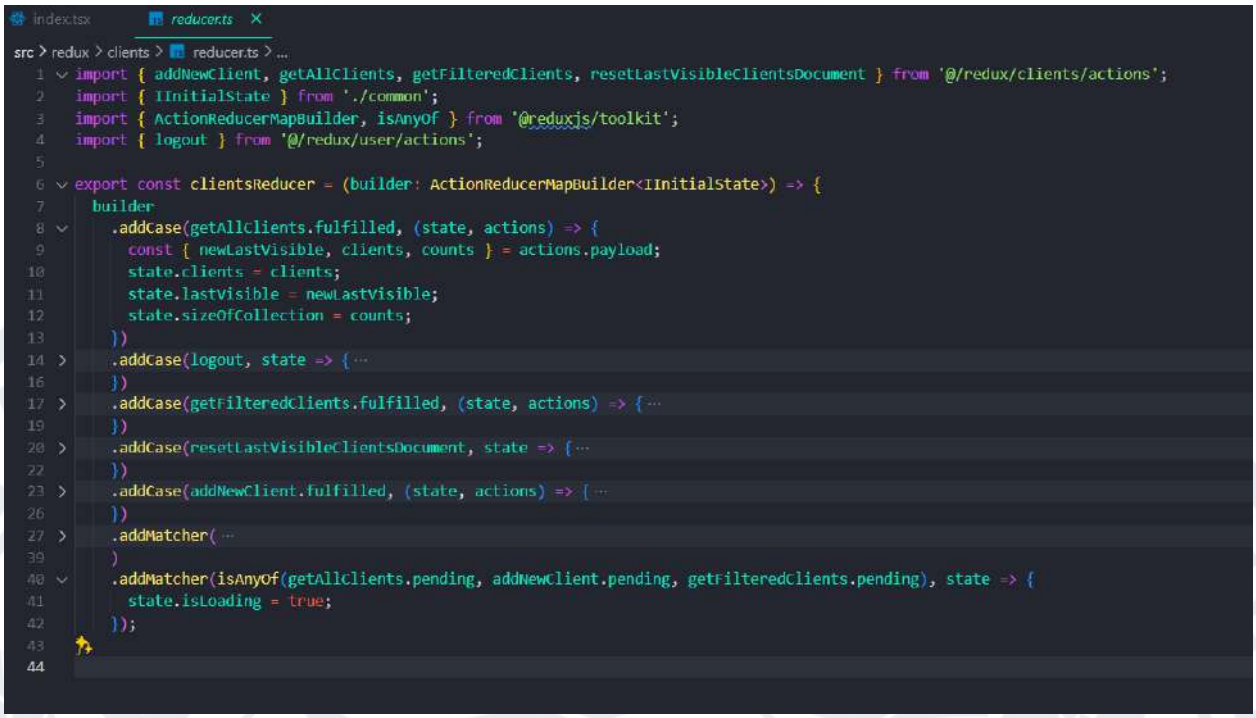


Рисунок 2.7 – Приклад Redux Reducer Клієнтів

Цей код представляє reducer для управління станом клієнтів у нашому фінансовому модулі. Він використовує Redux Toolkit для визначення reducer за допомогою методу builder з об'єкта ActionReducerMapBuilder.

Цей reducer відповідає за управління станом клієнтів у нашому модулі, включаючи отримання клієнтів, фільтрацію, додавання нових клієнтів та обробку стану завантаження. Його основна мета - забезпечити консистентність та ефективне керування станом додатку у відповідь на дії користувачів.

Узагальнюючи, Redux Toolkit відіграє важливу роль у спрощенні та оптимізації використання Redux у сучасних веб-застосунках, доповнюючи його Flux-подібну архітектуру та забезпечуючи потрібні інструменти для розробки складних веб-додатків.

РОЗДІЛ 3
РОЗРОБКА ФІНАНСОВОГО МОДУЛЮ

3.1 Робота з функціональною частиною «Клієнти» фінансового модулю

Колорист матиме змогу додати нового клієнта за допомогою модального вікна, вводитиме основні дані клієнта такі як його ПІБ, номер телефону, електронна пошта, адреса, марка машини, додаткова інформація, категорію клієнта. Категорій клієнтів ми маємо усього 3 – це VIP,Роздріб та Постійний Клієнт. Різниця між ними полягає в розміри знижки при наданні послуг. Деякі поля є необов'язковими, оскільки вони не є суттєво важливими, це такі поля як адреса, додаткова інформація та електронна пошта. Водночас такі поля як ПІБ та категорія клієнта є обов'язковими для вводу. Приклад вікна де представлено цю можливість наведено на рисунку 3.1

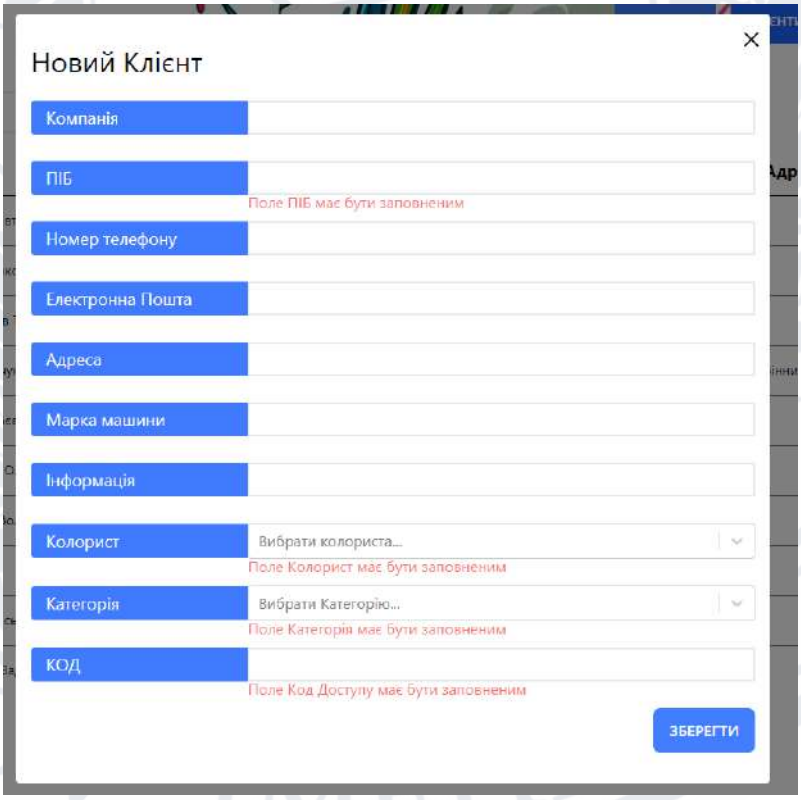


Рисунок 3.1 – Приклад модального вікна з додаванням нового клієнта

При введені усіх потрібних полів та при спробі зберегти дані, ми відправляємо HTTP запит на нашу серверну частину з новими даними про клієнта задля того, щоб зберегти усю інформацію в нашу базу даних.

Задля коректного збереження усіх даних полів, валідації цих полів та контролю їх я використала бібліотеку react hook form.

React Hook Form - це бібліотека для управління формами в React-додатках, яка зосереджується на використанні хуків (hooks). Вона призначена для спрощення процесу роботи з формами, забезпечуючи зручний та ефективний спосіб збирання, валідації та відправки даних. Це потужна та проста у використанні бібліотека для управління формами в React-додатках. Вона дозволяє швидко і легко створювати, валідувати та відправляти дані з форм, зменшуючи кількість необхідного коду і підтримуючи високу продуктивність та гнучкість.

Основні проблеми, які вирішує React Hook Form:

- Зменшення зайвого коду: React Hook Form дозволяє писати менше коду для реалізації форм, оскільки використовуються хуки та пропагація властивостей (prop spreading).
- Висока продуктивність: бібліотека оптимізована для використання хуків, що робить її дуже продуктивною та ефективною.
- Проста валідація: React Hook Form надає можливість використовувати різні методи валідації даних, такі як правила, встановлені користувачем, або вбудовані правила валідації.
- Управління станом: бібліотека дозволяє легко управляти станом форми та полями вводу, використовуючи хуки React.
- Форми без зберігання стану: React Hook Form дозволяє створювати форми без необхідності зберігання стану у власному стані компонента.

Основні переваги React Hook Form:

- Легкість використання: бібліотека має простий та зрозумілий API, що полегшує використання його навіть для початківців.

- Висока продуктивність: завдяки використанню хуків та оптимізації, React Hook Form працює швидко та ефективно навіть у великих додатках.

- Гнучкість: бібліотека дозволяє налаштовувати форми згідно з власними потребами та вимогами. Приклад використання даної бібліотеки наведено на рисунку 3.2

```
const {
  register,
  control,
  handleSubmit,
  formState: { errors },
} = useForm<IFormProps>({
  mode: 'all',
  defaultValues: {
    company: '',
    phoneNumber: '',
    email: '',
    address: '',
    carBrand: '',
  },
  resolver: joiResolver(addClientSchema),
});
```

Рисунок 3.2 – Приклад ініціалізації об’єкту React Hook Form для модального вікна зі створенням нового Клієнта

```
<Controller
  control={control}
  name="category"
  render={({ field: { value, onChange } }) => (
    <>
      <Select
        value={value}
        onChange={onChange}
        options={categoryOptions}
        className="w-full"
        placeholder="Вибрати категорію..."
        id="category"
      />
      <span className="text-md text-error absolute bottom-[-20px] left-0">
        {errors && errors.category && errors.category.message}
      </span>
    </>
  )}
/>
```

Рис 3.3 – Приклад використання React Hook Form в полі вибору категорії

Також, колорист отримуватиме повний список усіх його клієнтів отриманих з нашого серверу за допомогою того ж HTTP запиту. Приклад такого запиту наведено на рисунку 3.4

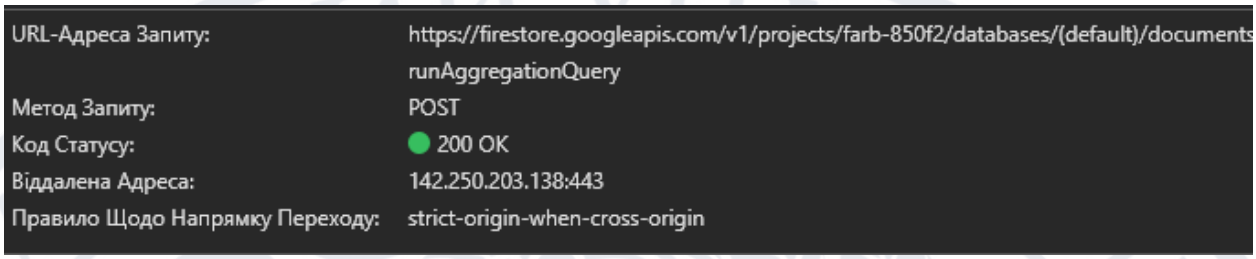


Рисунок 3.4 – Приклад запиту на сервер для отримання списку клієнтів

Ми обробляємо усі дані та подаємо їх у вигляді таблиці разом з пагінацією. Це зроблено заради того, щоб не перевантажувати клієнтську та серверну частину занадто великою кількістю даних. Взагалі, поняття пагінація це процес розділення великої кількості контенту на окремі сторінки з метою полегшення навігації для користувачів. Зазвичай пагінація використовується для відображення списків або наборів даних, таких як статті на блозі, продукти в інтернет-магазині або результати пошуку.

Основні елементи пагінації включають:

- Покажчики сторінок: елементи інтерфейсу, що дозволяють користувачам вибирати конкретну сторінку для перегляду.
- Кнопки «Назад» та «Вперед»: кнопки, які дозволяють користувачам переміщатися до попередньої або наступної сторінки.
- Інформація про кількість сторінок: інформація, яка вказує користувачам, скільки всього сторінок доступно для перегляду.
- Відображення кількості елементів на сторінці: інформація, що показує, скільки елементів відображається на кожній сторінці.
- Можливість переходу до певної сторінки: користувач може вводити номер сторінки в поле або обирати її зі списку для переходу.
- Пагінація дозволяє покращити досвід користувача, особливо коли маємо велику кількість даних. Вона допомагає уникнути завантаження всіх даних одразу, що

може призвести до зниження продуктивності додатка, і забезпечує зручний та ефективний спосіб навігації для користувачів. Приклад сторінки де продемонстровані дані можливості наведено на рисунку 3.5

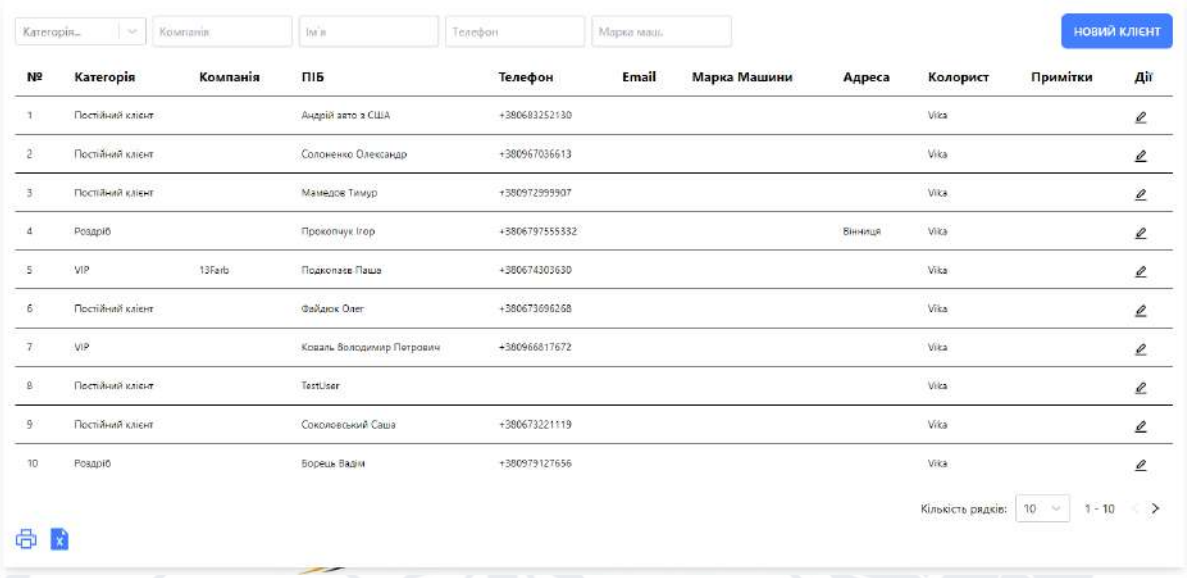


Рисунок 3.5 – Приклад сторінки зі список колористів та пагінацією

```
53    <div className="flex flex-col grow mt-3">
54      <>
55        <div className="overflow-x-auto w-full h-full grow">
56          <table className="table" ref={tableRef}>
57            <thead>
58              <tr>
59                <th className="text-black text-lg">№</th>
60                <th className="text-black text-lg">Категорія</th>
61                <th className="text-black text-lg">Компанія</th>
62                <th className="text-black text-lg">ПІБ</th>
63                <th className="text-black text-lg">Телефон</th>
64                <th className="text-black text-lg">Email</th>
65                <th className="text-black text-lg">Марка Машини</th>
66                <th className="text-black text-lg">Адреса</th>
67                <th className="text-black text-lg">Колорист</th>
68                <th className="text-black text-lg max-w-[180px]">Примітки</th>
69                <th className="text-black text-lg max-w-[180px]">Дії</th>
70              </tr>
71            </thead>
72            <tbody>
73              {filteredData.map(el => (···
99              ))}
100           </tbody>
101         </table>
102       </div>
```

Рис 3.6 – Приклад коду з таблицею усіх клієнтів

Під час відображення усіх даних, колорист також має зручну для себе змогу фільтрувати дані по певним полям. Наприклад по певній категорії клієнта, по його імені, телефону або ж по назві марки машини. Зроблено це заради

кращого User Experience. Приклад де продемонстровано даний функціонал надано на рисунку 3.7

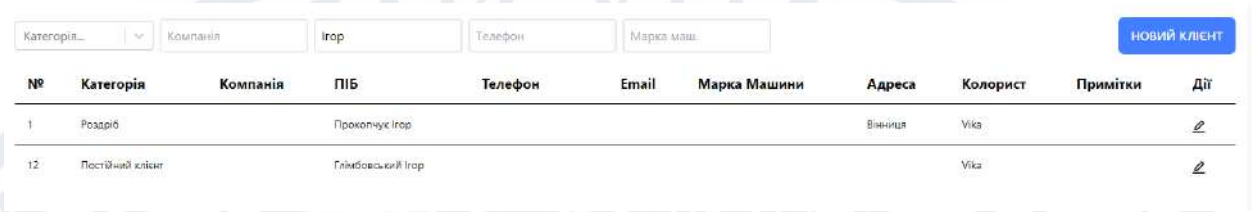


Рисунок 3.7 – Приклад Фільтрування Списку Клієнтів за іменем

Ще однією значущою можливістю є можливість редагування існуючого клієнта. Дуже часто виникає ситуація, коли певні дані про клієнта змінюються, і постійна актуальність цих даних є критично важливою. Для внесення змін в дані ми використовуємо аналогічне модальне вікно, як у процесі створення нового клієнта. Приклад вікна де можливо редагувати існуючого клієнта наведено на рисунку 3.8

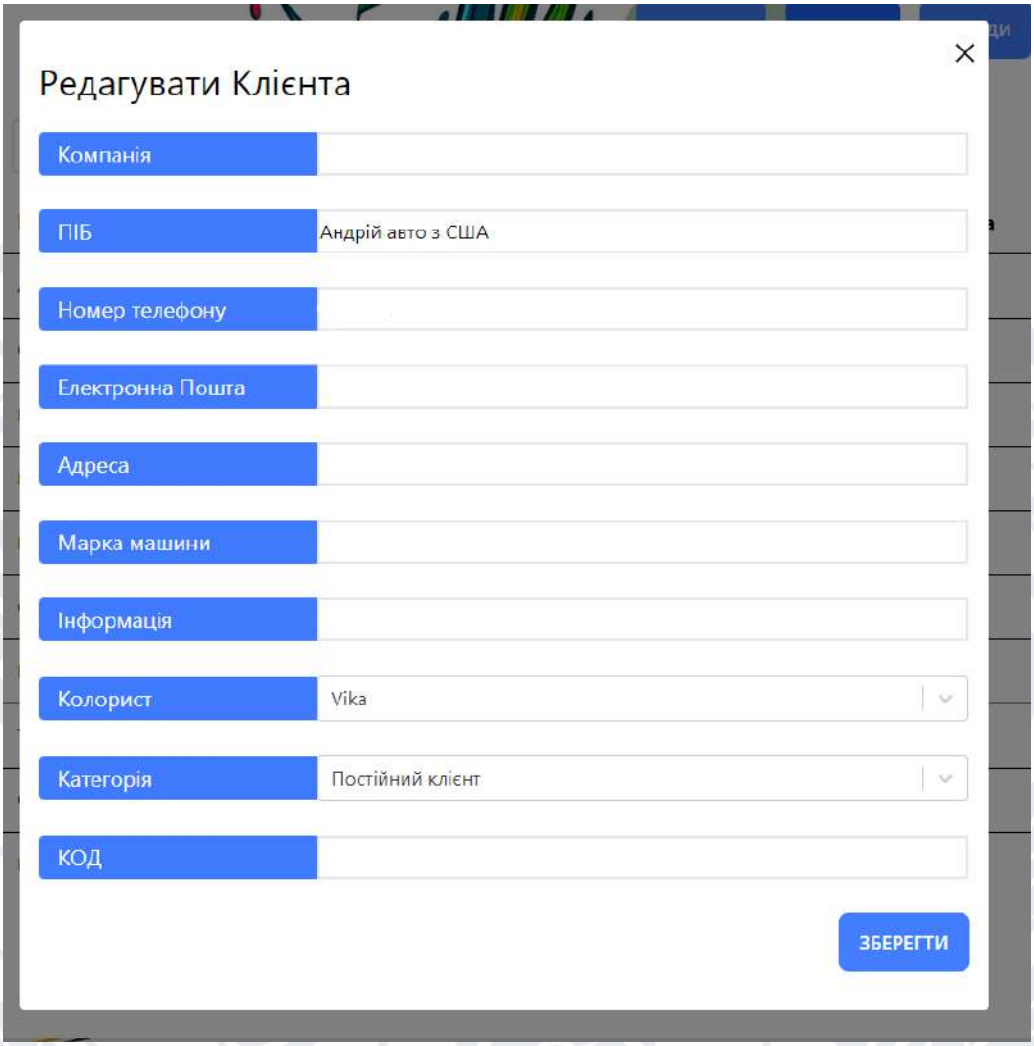


Рис 3.8 – Приклад Модального Вікна з редагуванням Клієнта

Надалі об’єкти клієнтів використовуватимуться в різних частинах усього застосунку.

3.2 Робота з функціональною частиною «Фінанси» фінансового модулю

Наш модуль має фінансовий аспект, що дозволяє редагувати основні змінні, які використовуються протягом роботи колориста. Ці змінні охоплюють різноманітні аспекти, такі як розмір знижки для різних категорій клієнтів (VIP, Роздріб, Постійний Клієнт), вартість послуги наливу за номером, вибір фарби за габаритами або авто та інші. Кожна змінна може бути змінена динамічно, що

спричиняє автоматичну адаптацію операцій, пов'язаних з клієнтами та фінансовими процесами, до нових значень змінних.

Усі зміни наших операцій фіксуються та надсилаються до нашого серверу для збереження нових значень. Приклад React компонента де представлені такі можливості наведено на рисунку 3.9

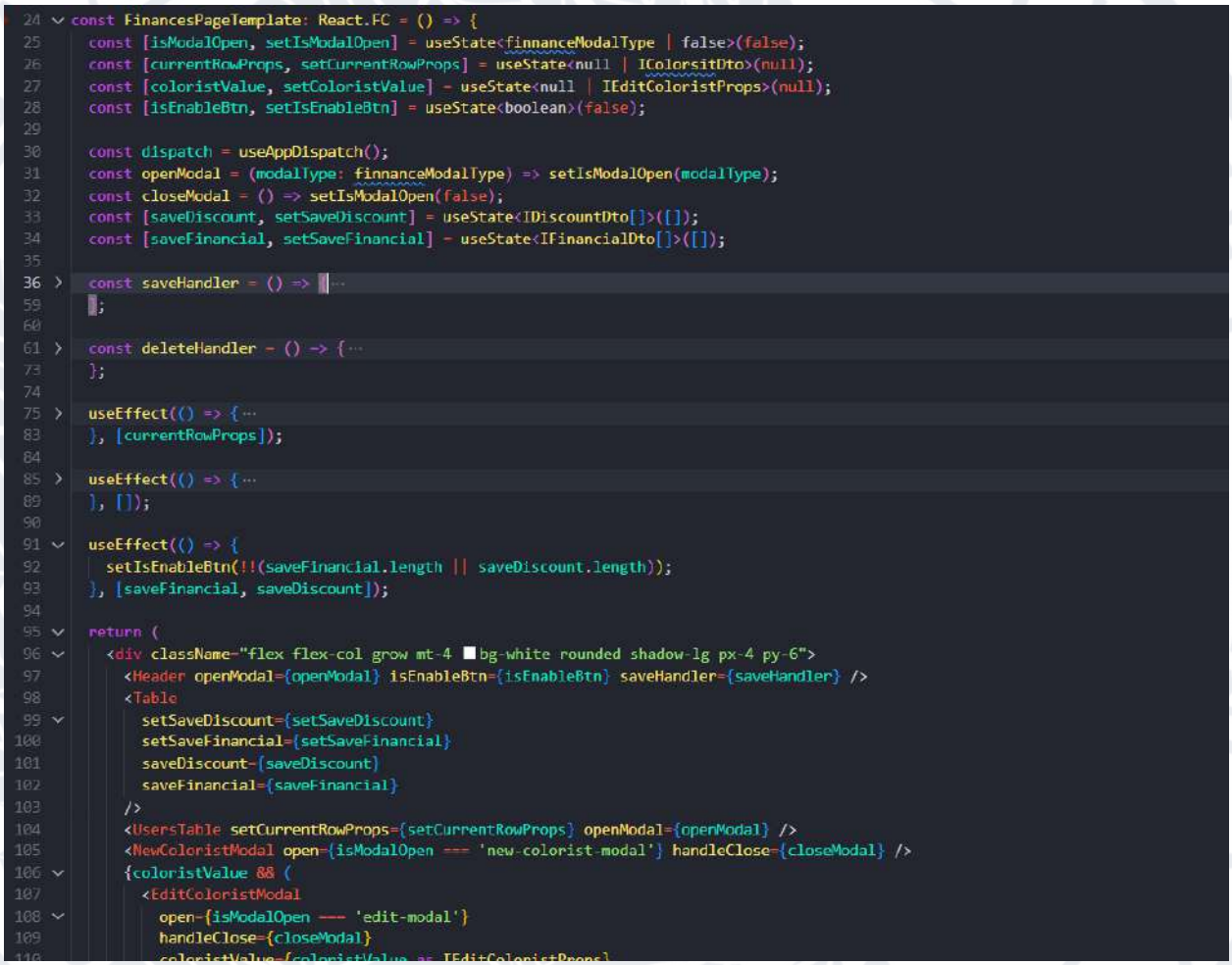


Рис 3.9 – Приклад React Компонента Фінансів

Також одним з ключових моментів даного блоку є можливість проводити повні CRUD операції над колористами. Загальне поняття CRUD операцій це - це скорочення від чотирьох базових операцій, які виконуються з даними у системі: Create (створення), Read (читання), Update (оновлення) та Delete (видалення). Ці операції є фундаментальними для будь-якої системи керування даними та використовуються для взаємодії з інформацією.

Create (створення): Ця операція використовується для створення нових записів або об'єктів у системі. Нові дані додаються до бази даних або іншого сховища даних. В нашому випадку це можливість додати нового колориста за потреби, якщо, наприклад, в компанії працюють декілька колористів.Від цього надалі буде залежати доволі суттєва частина функціоналу, оскількі ми матимемо можливість зазначати певні фінансові операції на конкретного колориста за допомогою його унікального коду. Приклад можливості зі створенням нового колориста наведено на рисунку 3.10

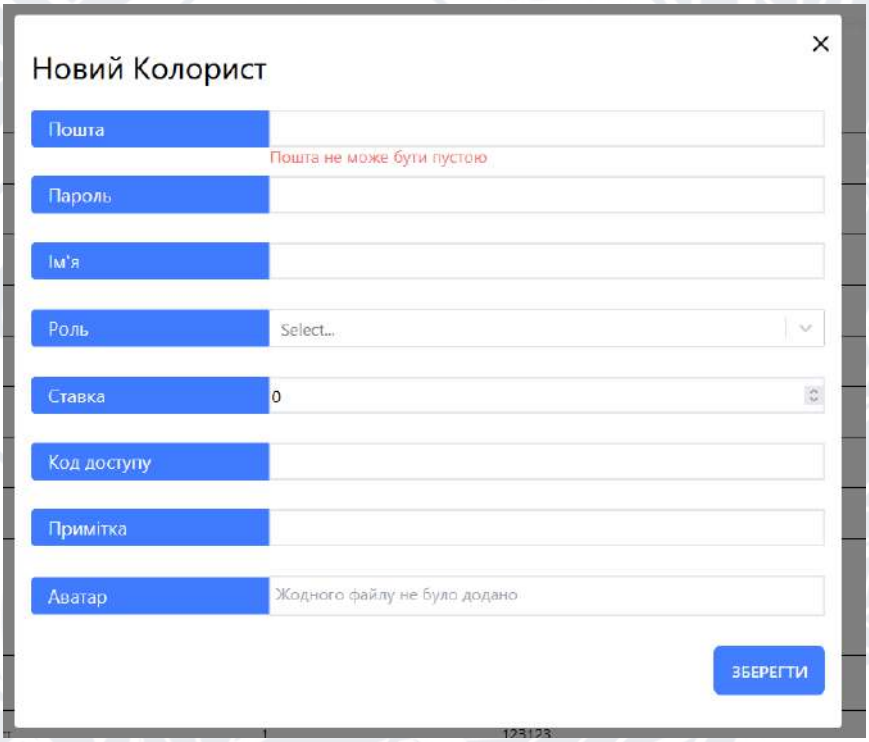


Рисунок 3.10 – Приклад Модального Вікна зі створенням нового колориста

Як можна побачити по даному малюнку, при створенні колориста необхідно ввести такі його дані як, пошта, пароль, ім'я, його роль, коефіцієнт ставки, код його доступу та аватар. Усі ці дані необхідні будуть новому колористу під час роботи з нашим застосунком та безпосередньо з фінансовим модулем, як наприклад для роботи з базою клієнтів, чи наприклад для звичайної авторизації.

Read (читання): Ця операція використовується для отримання інформації з бази даних або іншого сховища даних. Вона дозволяє отримувати інформацію про існуючі записи або об'єкти. Як приклад – це список усіх існуючих колористів відповідно нашої компанії, що доволі зручно бачити усіх співробітників в одному місці.

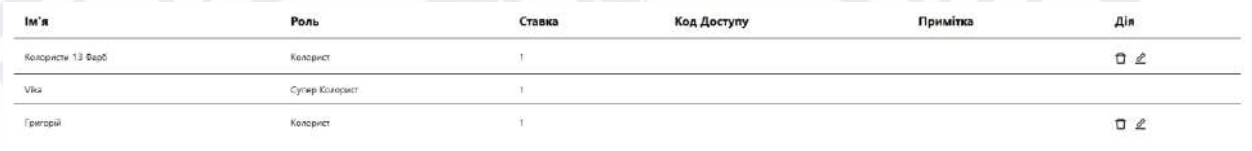


Рисунок 3.11 – Приклад UI блоку зі список колористів

Update (оновлення): Ця операція використовується для зміни існуючих записів або об'єктів у системі. Вона дозволяє оновлювати інформацію згідно з новими даними або вимогами. В нашому випадку це можливість редагувати існуючих колористів, змінювати їх код доступу, роль, пошту та пароль або ж аватар. Це зроблено задля того, щоб у разі зміни певних даних або ж їх втрати було легко їх замінити на нові.

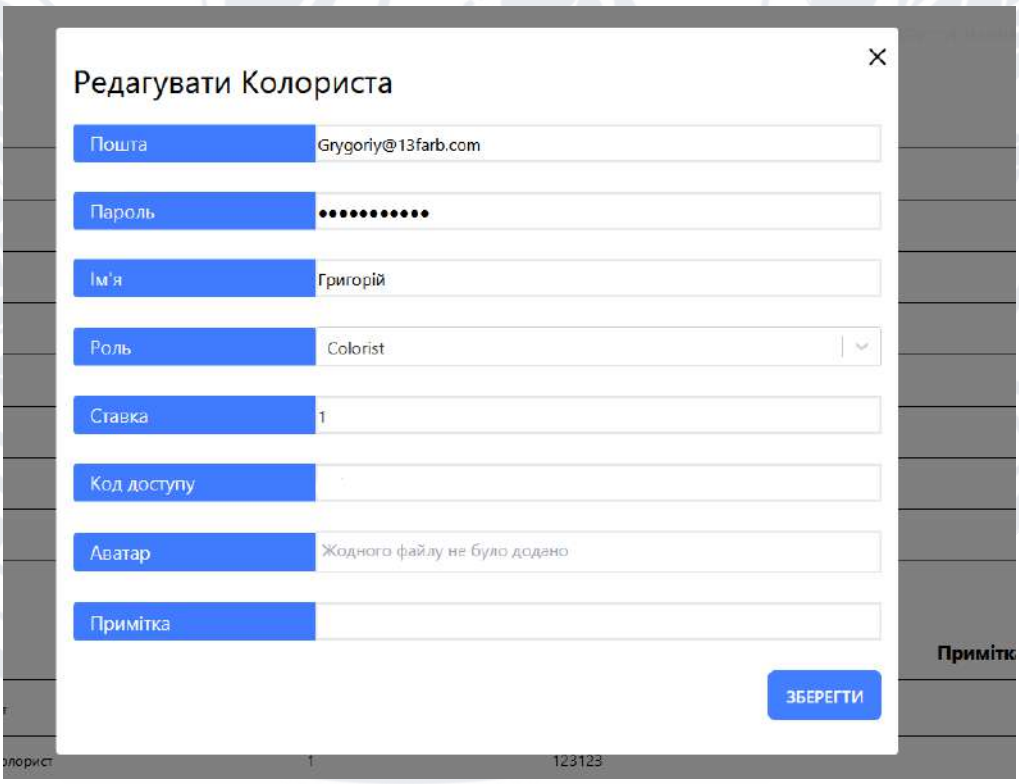


Рисунок 3.12 – Приклад Модального Вікна зміни даних колориста

Delete (видалення): Ця операція використовується для видалення існуючих записів або об'єктів з системи. Вона дозволяє видаляти непотрібну або застарілу інформацію. В нашому модулі колорист лише з роллю Супер Колорист має змогу видалити будь якого колориста. При цьому, при ініціалізації нашого застосунку ми обов’язково створюємо один акаунт супер – адміна, якого неможливо видалити. Це зроблено задля того, щоб в колориста був як мінімум один акаунтів з якого можна буде працювати з нашим застосунком.

Суть CRUD операцій полягає в тому, щоб надати повний контроль над даними у системі, дозволяючи створювати, зчитувати, оновлювати та видаляти їх за потребою. Це створює уніфікований та простий спосіб взаємодії з даними для користувачів та розробників систем. Завдяки цим операціям, можна ефективно керувати даними, забезпечуючи їхню цілісність та актуальність.

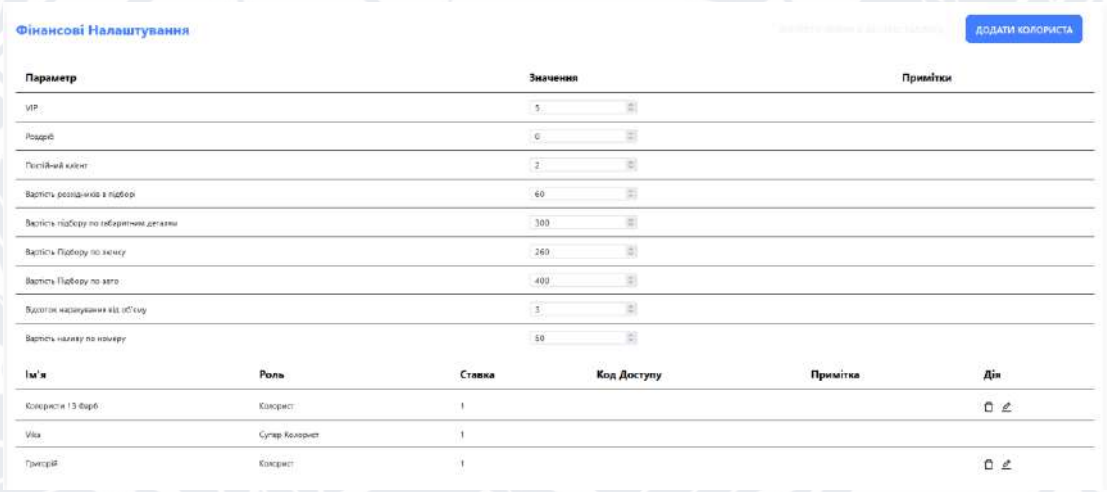


Рисунок 3.13 – Загальний Вигляд Сторінки Фінансові Налаштування

3.3 Імплементація Частини Підсумки Фінансового модулю

Заключною, проте не менш важливою частиною, є Підсумки нашого фінансового модулю. Розрахунок заробітної плати або ж чистого прибутку є невід’ємною частиною будь якого бізнесу, колористика не є виключенням. Відносно того, скільки операцій було проведено, на яку суму та яким колористом, визначається й формула отримання заробітної плати. Приклад

схеми по якій отримуються відповідні суми заробітньої плати наведено на рисунку 3.14

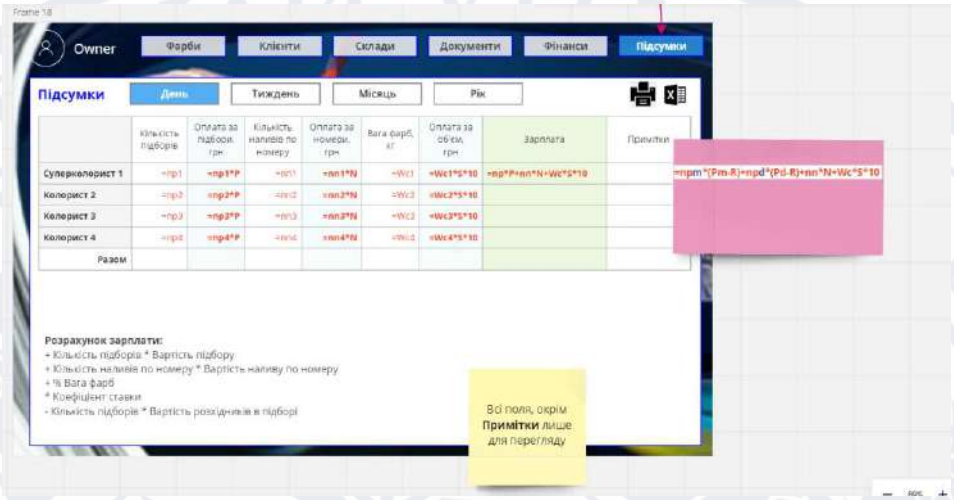


Рисунок 3.14 – Схема прорахунку заробітної плати кожного колориста в фінансовому модулі

Зі схеми для розрахунку використовуються різні змінні, включаючи кількість підборів, кількість наливів за номером, коефіцієнт ставки та кількість підборів. Усі ці змінні отримуються з нашого серверу в результаті взаємодії з фінансовим модулем та іншими компонентами системи. Важливо зауважити, що ці змінні допомагають у прорахунку заробітної плати та інших фінансових показників, що визначаються в процесі роботи з системою. Приклад з кодом зі змінними наведено на рисунку 3.15

```
29    const PA = useAppSelector(selectFinancialPriceOfSelectByCar);
30    const PD = useAppSelector(selectFinancialPriceOfSelectByDetail);
31    const PHD = useAppSelector(selectFinancialPriceOfSelectByHugeDetail);
32    const N = useAppSelector(selectFinancialPriceOfNumber);
33    const S = useAppSelector(selectFinancialPercentOfCapacity);
34    const R = useAppSelector(selectFinancialPriceOfSpendthrifts);
35
36  > useEffect(() => {···
40    }, [colorists, startDate, endDate]);
41
42    useEffect(() => {
43      setTableDataToExcel(summaryTableDataTranfer(PA, PD, S, N, filteredData, R));
44    }, [filteredData]);
45
46    if (isLoading) return <span className="loading loading-spinner loading-lg"></span>;
```

Рисунок 3.15 – Приклад Коду з отриманими змінними

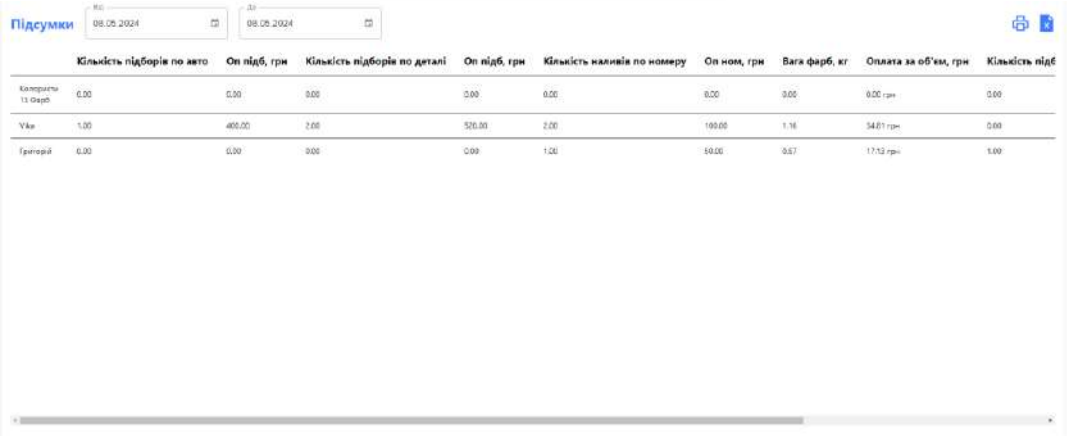


Рисунок 3.16 – Приклад Сторінки Підсумки

Колорист також має змогу не лише отримати повний список прорахунку заробітної плати, а й налаштовувати період за який будуть відображатися відповідні дані, це зручним тим, що колорист не завжди бажає бачити результати заробітної плати за лише один період, а заради зручності та звітності є можливість обирати різні періоди.

Під час вибору конкретного періоду, наш код ловить цей момент та відправляє запит до серверу де той згодом повертає відповідь з даними, які ми зберігаємо згодом в нашому Redux сховищі. Приклад вибору такого періоду наведено на рисунку 3.17

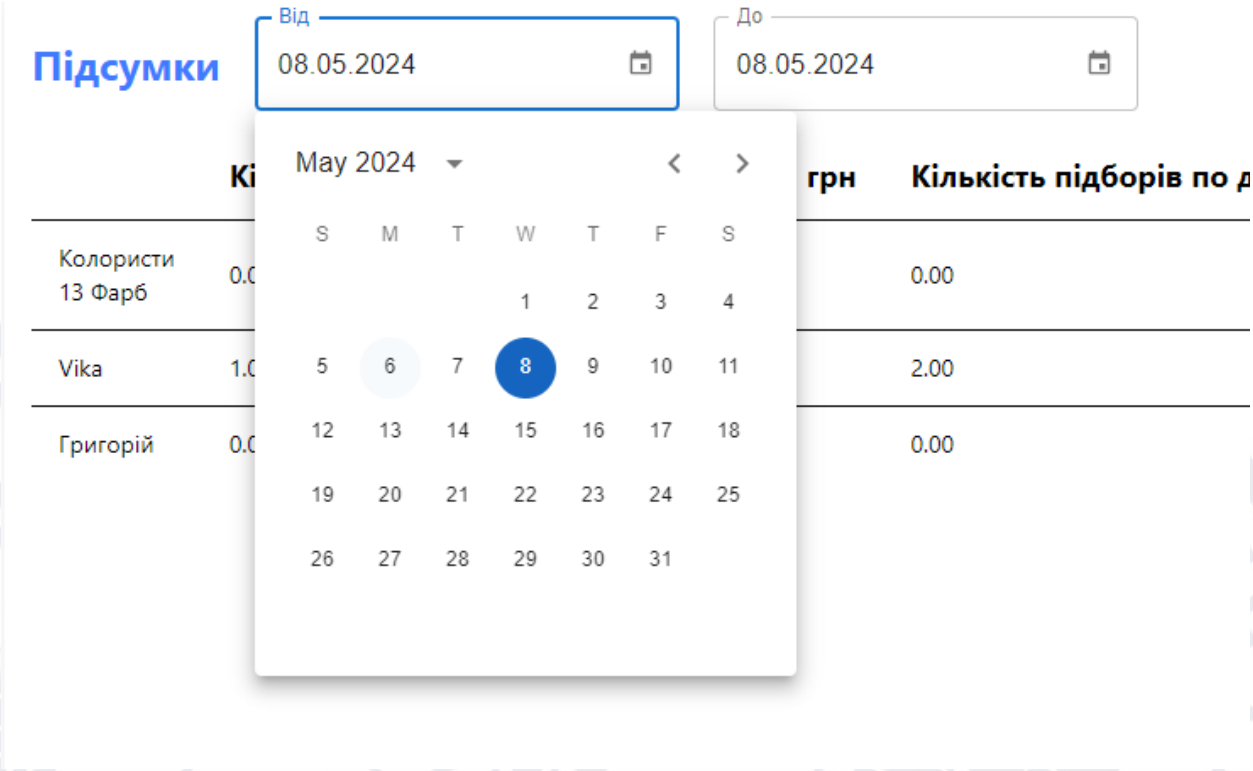


Рисунок 3.17 – Приклад Вибору Дати В Частині Підсумки Фінансового Модулю

3.4 Функціонал друку даних та виведення їх в Excel таблиці

Задіяння друкованих таблиць та експорту даних у формат Excel є важливою функціональністю для будь яких Фінансових модулів. Ці можливості дозволяють користувачам зручно та ефективно використовувати зібрані дані для подальшого аналізу, звітності та прийняття рішень.

В модулі було додано таку можливість звітності, яка дозволяє користувачам легко друкувати таблиці, які вони переглядають на сторінці, а також експортувати ці дані безпосередньо у формат Excel. Це надає можливість збереження важливих інформаційних звітів, аналізувати їх в зручному середовищі та ділитися ними з іншими зацікавленими сторонами.

Під час написання кодової частини цього завдання я використала таку бібліотеку як “react-to-print”, яка дозволяє легко та зручно керувати можливістю друку певних HTML елементів.

```
<ReactToPrint
  bodyClass="print-agreement"
  content={() => tableRef.current}
  trigger={() => <PrinterOutlined className="text-3xl text-primary leading-[0] cursor-pointer hidden-on-print" />}
/>
```

Рисунок 3.18 – Приклад використання бібліотеки react-to-print для друку таблиці клієнтів

Для перенесення даних в Excel таблиці я використала бібліотеку react-csv, API якої легко налаштовується та інтегрується в наш застосунок

```
22      <CSVLink data={excelData} headers={headers} filename={fileName} separator={','}>
23        <FileExcelFilled className="text-3xl text-primary leading-[0] cursor-pointer hidden-on-print" />
24      </CSVLink>
```

Рисунок 3.19 – Приклад використання бібліотеки react-csv для отримання Excel таблиці клієнтів

Основний функціонал друку таблиці заключається у тому, що користувач при натисканні на кнопку друку, бачить перед собою приклад вигляду роздрукованої таблиці в чорнобілому варіанті, після того як колорист переконався у вірності друку даних, він відправляє відповідну сторінку на друк та отримує фізичний приклад своєї таблиці. Це значно спрощує та пришдвишує процес фінансової звітності та надає можливотсі порівнювати результати відносно певного періоду. А також зберігати фізично відповідні дані у разі неможливості доступу до модулю. Приклад можливості друку наведено на рисунку 3.20

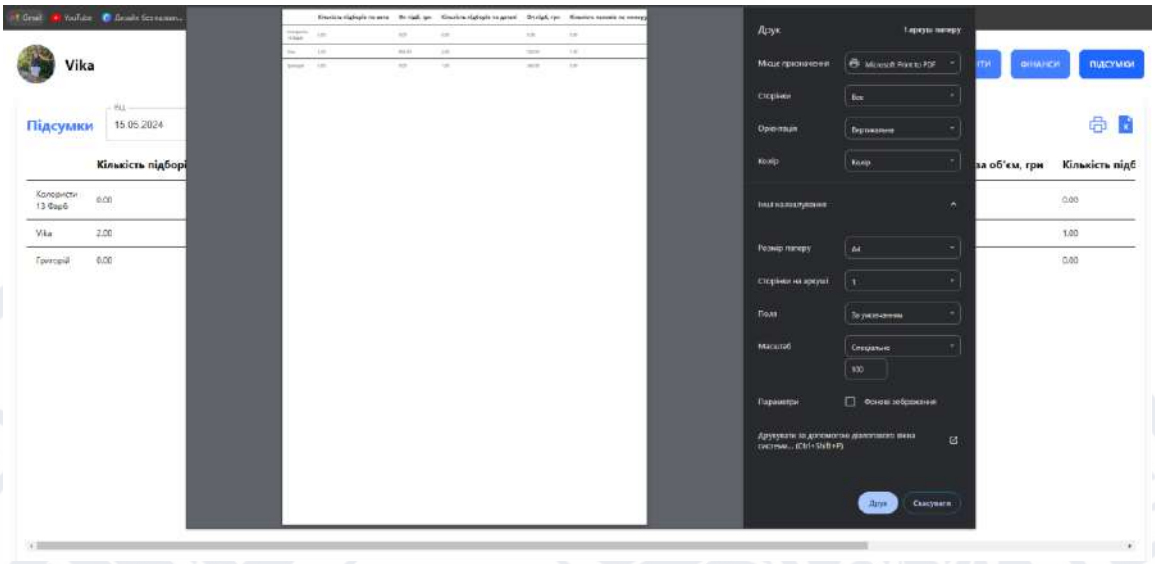
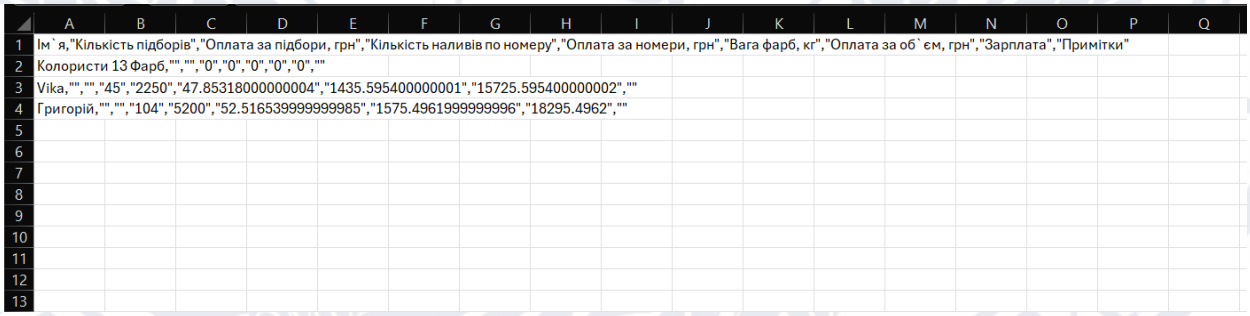


Рисунок 3.20 – Вікно налаштування друку таблиці підсумків заробітної плати за певний проміжок часу

Перенесення вигляду таблиці у Excel формат також зручний для збереження звітності та подальшої інтеграції з будь якими новітніми технологіями та методами Excel. Це дозволяє легко та зручно зберігати таблиці та дані у відповідному Excel файлу. Приклад отриманої таблиції Excel наведено на рисунку 3.21



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	Ім`я,"Кількість підборів","Оплата за підбори, грн","Кількість наливів по номеру","Оплата за номери, грн","Вага фарб, кг","Оплата за об`єм, грн","Зарплата","Примітки"																
2	Колористи 13 Фарб,"","","0","0","0","0",""																
3	Vika,"","","45","2250","47.85318000000004","1435.595400000001","15725.595400000002",""																
4	Григорій,"","","104","5200","52.516539999999985","1575.4961999999996","18295.4962",""																
5																	
6																	
7																	
8																	
9																	
10																	
11																	
12																	
13																	

Рисунок 3.21 – Приклад Excel таблиці даних заробітної плати за певний проміжок часу

3.5 Реалізація автотестів з використанням Cypress

Написання автотестів для покриття ними застосунків в сучасних веб додатках є важливою та невід’ємною частиною процесу розробки якісних та

новітніх програм, наш фінансовий модуль не є виключенням. Під час написання автотестів, я ставила за мету перевірити коректність роботи функціоналу фінансового модулю, а саме роботу частини з внесенням даних клієнтів, їх редагування та створення. Для досягнення цієї мети я розробила тестові сценарії, які перевіряють наявність необхідних даних, їх відображення та правильність сортування та фільтрації результатів.

Для перевірки функціоналу збереження, створення та редагування клієнтів у веб-інтерфейсі, я розробила обширний набір E2E тестових кейсів у Cypress. Ці тести не лише симулюють дії користувача на веб-сайті, але і переконуються в правильності введення нових даних у форми, їхньому збереженні в базі даних та можливості редагування існуючих записів.

Кожен тестовий кейс у цьому наборі ретельно розроблений для відтворення реальних сценаріїв взаємодії з клієнтською частиною програми. Наприклад, один із кейсів може включати створення нового клієнта, заповнення усіх відповідних полів у формі, натискання кнопки "Зберегти", а потім перевірку на навявність відповідних даних в нашій таблиці. Приклад написаних автотестів наведено на рисунку 3.22

```
describe('Client Page Tests', () => {
  beforeEach(() => {
    cy.viewport(1920, 1080);
    cy.visit('http://localhost:5173/auth');
    cy.get("input[name='email']").type('testuser@gmail.com');
    cy.get("input[name='password']").type('123456');
    cy.get('button').click();
    cy.get('button:contains("Клієнти")').click();
  });

  it('User after clicking on tab Clients should be redirected to clients page', () => {
    cy.url().should('contain', '/clients');
  });
  it('After click on button New Client there should be pop up', () => {
    cy.get('button:contains("Новий Клієнт")').click();
    cy.get('p:contains("Новий Клієнт")').should('exist');
  });

  it('If All Data in Form is valid new user should be created', () => {
    let count = 0;
    cy.get('table tr')
      .its('length')
      .then(val => {
        count = val;
        cy.get('button:contains("Новий Клієнт")').click();
        cy.get("input[name='fio']").type('Тестовий Юзер');
        cy.get("div[id='colorist']").click();
        cy.get("div[class*='option']").first().click();
        cy.get("div[id='category']").click();
        cy.get("div[class*='option']").first().click();
        cy.get("input[name='accessCode']").type('123123');
        cy.get("button:contains('Зберегти')").click();
      });
    cy.wait(4000);
```

Рисунок 3.22 – Приклад автотестів для сторінки “Клієнти” фінансового модулю

Під час написання даних автотестів я використала підхід написання End-to-End тестів. End to End (E2E) тести в Cypress - це тестування програмного забезпечення, яке охоплює всі аспекти програми від початку до кінця, симулюючи реальні дії користувача. Ці тести перевіряють, як різні компоненти програми взаємодіють між собою в реальному середовищі, від взаємодії із зовнішніми API до взаємодії з іншими складовими програми.

Перевагами використання даного підходу є

- Виявлення проблем на ранніх етапах розробки: E2E тести допомагають виявити проблеми та помилки ще до випуску продукту.
- Стабільність програми: Широке покриття тестами дозволяє виявити та усунути багато проблем, що можуть виникнути при взаємодії різних компонентів програми.
- Зменшення ризику регресії: E2E тести допомагають запобігти появі регресійних помилок при внесенні змін у вихідний код.

- Покращення якості продукту: Вони допомагають забезпечити високу якість продукту шляхом виявлення і усунення помилок та недоліків.

- Ефективне використання ресурсів: Автоматизація E2E тестів зменшує необхідність в ручному тестуванні та збільшує ефективність розробки програмного забезпечення.

Загалом, E2E тести в Cypress є важливою складовою процесу тестування програмного забезпечення, яка допомагає забезпечити якість, надійність та стабільність програми перед її випуском. Приклад використання автотестів Cypress наведено на рисунку 3.23

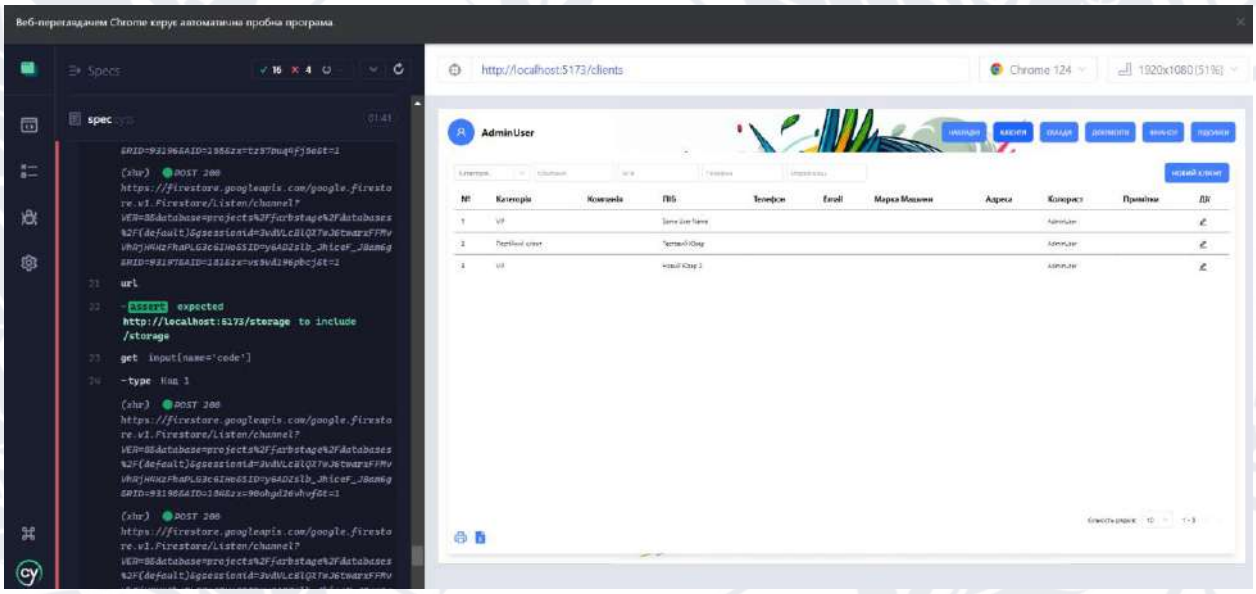


Рисунок 3.23 – Приклад виконання автотестів на сторінці клієнти

ВИСНОВКИ

У ході виконання даної роботи були успішно досягнуті поставлені завдання, що спрямовані на підвищення ефективності та оптимізацію робочих процесів колористів в автомобільній галузі. Однією з ключових досягнутих мет є розробка фінансового модулю для CRM системи, який був спеціально адаптований до унікальних потреб колористів. Цей модуль надає зручні та ефективні інструменти для керування обліком клієнтів, налаштування ключових параметрів у фінансових операціях та автоматизації процесу отримання коректних фінансових звітів за визначений період часу.

У рамках дослідження було ретельно проаналізовано поняття колористів, їхню специфіку роботи та виявлено основні потреби у фінансовому модулі. Додатково був проведений аналіз ринку для оцінки наявності аналогічних рішень, що підтвердив актуальність та необхідність розробки спеціалізованого фінансового інструменту для колористів.

Під час вибору технологічних рішень були ретельно розглянуті та проаналізовані сучасні веб-технології з метою вибору оптимальних інструментів, які відповідають поставленим цілям та вимогам проекту. Такий підхід дозволив вибрати технології, які найкращим чином відповідають потребам колористів і забезпечують ефективну та стабільну роботу фінансового модулю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Офіційна документація **React**. URL: https://react.dev/
2. Getting started with React стаття URL: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started
3. An introduction to the Flux architectural pattern стаття URL: https://www.freecodecamp.org/news/an-introduction-to-the-flux-architectural-pattern-674ea74775c9/
4. What is Flux? стаття URL: http://fluxxor.com/what-is-flux.html
5. Офіційна документація Redux Toolkit URL: https://redux-toolkit.js.org/
6. Redux Toolkit як засіб ефективної розробки стаття URL: https://markup-ua.com/redux-toolkit-yak-zasib-efektivno%D1%97-rozrobki/
7. Використання Redux з React: Кращі практики стаття URL: https://medium.com/@front3nd/%D0%B2%D0%B8%D0%BA%D0%BE%D1%80%D0%B8%D1%81%D1%82%D0%B0%D0%BD%D0%BD%D1%8F-redux-%D0%B7-react-%D0%BA%D1%80%D0%B0%D1%89%D1%96-%D0%BF%D1%80%D0%B0%D0%BA%D1%82%D0%B8%D0%BA%D0%B8-e5f88bcfd84e
8. Офіційна документація Cypress URL: https://www.cypress.io/
9. Ніколи не писали автотести? Спробуйте Cypress стаття URL: https://dou.ua/forums/topic/31487/
10. End-to-End тестування в деталях та прикладах стаття URL: https://foxminded.ua/end-to-end-testuvannia/
11. Що таке End-to-End тестування стаття URL: https://www.it-notes.wiki/software-testing/what-is-end-to-end-testing/
12. Vite — наступник WebPack? стаття URL: https://medium.com/@jstify.community/vite-%D0%BD%D0%B0%D1%81%D1%82%D1%83%D0%BF%D0%BD%D0%B8%D0%BA-webpack-9f2fdc847c79
13. Використання Vite для швидкого розгортання веб-додатків на основі Vue та React стаття URL:

https://it-rating.ua/vikoristannya-vite-dlya-shvidkogo-rozgortannya-veb-dodatkiv-na-osnovi-vue-ta-react

14. Офіційна Документація Vite URL: https://vitejs.dev/

ДОДАТОК А

Лістинг React компоненту сторінки «Підсумки»

```tsx
import React, { useEffect, useRef, useState } from 'react';
import Header from './components/Header';
import Table from './components/Table';
import { useAppDispatch } from '@/redux/hooks';
import { getAllColorists } from '@/redux/colorists/actions';
import { getFinancilaCategories } from '@/redux/finances/actions';
import { ISummaryExcelTableColoristDto } from '@/types/interface';
import dayjs, { Dayjs } from 'dayjs';

const SummariesPageTemplate: React.FC = () => {
  const dispatch = useAppDispatch();
  const tableRef = useRef<HTMLTableElement | null>(null);
  const [tableDataToExcel, setTableDataToExcel] = useState<ISummaryExcelTableColoristDto[]>([]);
  const [startDate, setStartDate] = useState<Dayjs | null>(dayjs());
  const [endDate, setEndDate] = useState<Dayjs | null>(dayjs());

  useEffect(() => {
    dispatch(getAllColorists());
    dispatch(getFinancilaCategories());
  }, []);

  return (
    <div className="flex flex-col grow mt-4 bg-white rounded shadow-lg px-4 py-6">
      <Header
        startDate={startDate}
        setStartDate={setStartDate}
        endDate={endDate}
        setEndDate={setEndDate}
        tableRef={tableRef}
        tableDataToExcel={tableDataToExcel}
      />
      <Table tableRef={tableRef} setTableDataToExcel={setTableDataToExcel} startDate={startDate} endDate={endDate} />
    </div>
  );
};

export default React.memo(SummariesPageTemplate);
```

ДОДАТОК Б

Лістинг Cypress E2E тестів фінансового модулю

```
describe('Authorization Tests', () => {
  beforeEach(() => {
    cy.viewport(1920, 1080);
    // cy.visit('https://farbstage.web.app/auth');
    cy.visit('http://localhost:5173/auth');
  });
  it('Users with valid data should be redirected to the invoces page', () => {
    cy.get("input[name='email']").type('testuser@gmail.com');
    cy.get("input[name='password']").type('123456');
    cy.get('button').click();
    cy.url().should('contain', '/invoces');
  });
  it('Users with invalid data shouldn`t be redirected and should view the message', () => {
    cy.get("input[name='email']").type('ggg@gmail.com');
    cy.get("input[name='password']").type('123456');
    cy.get('button').click();
    cy.url().should('contain', '/auth');
    cy.get('div.Toastify__toast-container').should('exist');
  });
  it('Enter Button shouldn`t be disable when there is password and email', () => {
    cy.get("input[name='email']").type('ggg@gmail.com');
    cy.get("input[name='password']").type('123456');
    cy.get('button').should('not.be.disabled');
  });
  it('Enter Button should be disabled when there is no email or password', () => {
    cy.get("input[name='password']").type('123456');
    cy.get('button').should('be.disabled');
  });
});

describe('Client Page Tests', () => {
  beforeEach(() => {
    cy.viewport(1920, 1080);
    cy.visit('http://localhost:5173/auth');
    cy.get("input[name='email']").type('testuser@gmail.com');
    cy.get("input[name='password']").type('123456');
    cy.get('button').click();
    cy.get('button:contains("Клієнти")').click();
  });

  it('User after clicking on tab Clients should be redirected to clients page', () => {
    cy.url().should('contain', '/clients');
  });
```

```
  it('After click on button New Client there should be pop up', () => {
    cy.get('button:contains("Новий Клієнт")').click();
    cy.get('p:contains("Новий Клієнт")').should('exist');
  });

  it('If All Data in Form is valid new user should be created', () => {
    let count = 0;
    cy.get('table tr')
      .its('length')
      .then(val => {
        count = val;
        cy.get('button:contains("Новий Клієнт")').click();
        cy.get("input[name='fio']").type('Тестовий Юзер');
        cy.get("div[id='colorist']").click();
        cy.get("div[class*='option']").first().click();
        cy.get("div[id='category']").click();
        cy.get("div[class*='option']").first().click();
        cy.get("input[name='accessCode']").type('123123');
        cy.get("button:contains('Зберегти')").click();
      });
    cy.wait(4000);
    cy.get('table tr')
      .its('length')
      .then(newVal => {
        expect(newVal).to.be.eq(count + 1);
      });
  });
  it('If we try to save the form without any data there is should be a few error messages', () => {
    cy.get('button:contains("Новий Клієнт")').click();
    cy.get("button:contains('Зберегти')").click();
    cy.get('span:contains("Поле ПІБ має бути заповненим")').should('exist');
    cy.get('span:contains("Поле Колорист має бути заповненим")').should('exist');
    cy.get('span:contains("Поле Категорія має бути заповненим")').should('exist');
    cy.get('span:contains("Поле Код Доступу має бути заповненим")').should('exist');
  });
  it('If we select first option in colorists selector the value should be first option', () => {
    let optionText = '';
    cy.get('button:contains("Новий Клієнт")').click();
    cy.get("div[id='colorist']").click();
    cy.get("div[class*='option']")
      .first()
      .invoke('text')
```

ДОДАТОК В

Структура сторінок «Фінанси» та «Підсумки» Фінансового модулю