Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБКА ТА ВПРОВАДЖЕННЯ СИСТЕМИ УПРАВЛІННЯ РАХУНКАМИ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-20-2
Шандра В. Р.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Шафроненко А. Ю.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Шандрі Владі Ростиславівні
(прізвище, ім'я, по батькові)

1. Тема роботи Розробка та впровадження системи управління рахунками

затверджена наказом університету від 20 травня 2024 року № 464 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 28 травня 2024 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, дані інтернет-мережі.

4. Перелік питань, що потрібно опрацювати в роботі

1. Аналіз предметної галузі та постановка задачі.
2. Аналіз специфікації вимог та створення інтерфейсу системи для управління рахунками.
3. Комп'ютерна модель системи управління рахунками.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність завдання створення системи управління рахунками, технологічні рішення при реалізації вебзастосунку, інтерфейс користувача програми

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	08.04.2024	
2	Аналіз завдання, підбір літератури	08.04.24-12.04.24	
3	Аналіз літератури з досліджуваної проблеми	13.04.24-16.04.24	
4	Аналіз технічних засобів	17.04.24-20.04.24	
5	Розробка методу	21.04.24-08.05.24	
6	Програмна реалізація	09.05.24-20.05.24	
7	Оформлення пояснювальної записки	21.05.24-24.05.24	
8	Перевірка на плагіат	27.05.24	
9	Рецензування	28.05.24	
10	Підготовка презентації та доповіді	29.05.24-06.06.24	
11	Занесення роботи в електронний архів	06.06.24	
12	Попередній захист кваліфікаційної роботи	06.06.24	

Дата видачі завдання 8 квітня 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Шафроненко А.Ю._____
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи: 64 с., 5 табл., 47 рис., 37 джерел.

ВЕБЗАСТОСУНОК, РАХУНКИ, БАЗА ДАНИХ, SQL HTML, CSS, JavaScript, REACT, NODE.JS, NEXT.JS.

Об’єктом роботи є система управління рахунками.

Метою роботи є розробка системи управління рахунками для електронної комерційної діяльності бізнесу.

Під час розробки застосунку було використано такі технології: мова програмування JavaScript; серверне середовище виконання JavaScript Node.js; фреймворк розробки застосунків React Next.js; база даних PostgreSQL, мова запитів SQL; середовище розробки програмних систем MSVisualStudioCode; програма для проєктування інтерфейсу та створення прототипів Figma; інструмент керування базою даних PostgreSQL pgAdmin; об’єктно-орієнтованої служби зберігання Amazon S3.

У результаті роботи здійснена програмна реалізація застосунку для управління рахунками.

WEB APPLICATION, INVOICE, DATABASE, SQL HTML, CSS, JavaScript, REACT, NODE.JS, NEXT.JS.

The object of the work is the account management system.

The purpose of the work is to develop an account management system for electronic commercial activities of businesses.

During the development of the application, the following technologies were used: JavaScript programming language; JavaScript server execution environment Node.js; React Next.js application development framework; PostgreSQL database, SQL query language; MSVisualStudioCode development environment for software systems; program for interface design and prototyping Figma; pgAdmin PostgreSQL database management tool; object-oriented storage service Amazon S3.

As a result of the work, the software implementation of the invoicing application was carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

БД − база даних

ПДВ – податок на додану вартість

EDI – Electronic Data Interchange (система електронного обміну даними)

HTML − HyperText Markup Language (мова розмітки гіпертексту)

SQL − Structured Query Language (мова структурованих запитів)

CSS − Cascading Style Sheets (каскадні таблиці стилів)

AWS − Amazon Web Services

HTTP − HyperText Transfer Protocol (протокол передачі даних)

ORM − Object-Relational Mapping (об’єктно-реляційна проєкція)

ERD – діаграма взаємозв’язку сутностей

API – Application Programming Interface (програмний інтерфейс застосунку)

SSG – Static Site Generation (генерація статичних сайтів

SSR − Server-Side Rendering (серверна генерація сторінок)

JSON − JavaScript Object Notation (текстовий формат обміну даними між комп’ютерами)

ВСТУП

За останні 10 років ринок зазнав значних змін у зв’язку з розвитком технологій у світі, не останньою з яких є стрімке зростання електронної комерції. У 2010 році менше 5% роздрібних продажів відбувалося в Інтернеті. За це десятиліття зростання електронної комерції ця цифра зросла більш ніж утричі – до 18%.

Пандемія COVID-19 також мала величезний вплив. З цієї причини більша частина цього зростання відбулася у 2020 році, коли продажі електронної комерції зросли на 44%. З 2010 року глобальні продажі електронної комерції зросли майже на 800% [1].

Український ринок електронної комерції продовжує зростати. У 2023 році його обсяг склав 151 млрд грн, що на 17% більше, ніж у 2022 році [2]. Цей тренд, ймовірно, продовжиться і у 2024 році, і швидше за все, зростання навіть пришвидшиться. Очевидно, що конкуренція в сфері електронної комерції також зростає з року в рік.

Тому компаніям потрібні нові застосунки, які дозволять швидко реагувати на зміни ринку та здійснювати операції з продажу, а також відстежувати потенційний прибуток.

В рамках цієї роботи створено програму для автоматизації процесів, пов’язаних із формуванням рахунків.

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальні відомості з предметної галузі

Підприємницька діяльність – це комерційна діяльність, яка спрямована на виробництво, купівлю-продаж, або надання послуг з метою отримання прибутку. Вона відіграє важливу роль у стимулюванні економічного зростання та розвитку, допомагаючи створювати нові робочі місця, сприяти інноваціям та підтримувати конкуренцію на ринку [3].

Створення рахунків в підприємницькій діяльності є процесом фіксації здійснених операцій продажу товарів або послуг. Поява рахунків в грошових відносинах було необхідним для забезпечення точного обліку та документації фінансових операцій. Вони надають запис про те, що було продано, кому це було продано та дату продажу. Рахунки допомагають забезпечити точність фінансових записів, полегшують відстеження взаєморозрахунків та допомагають управляти готівковими потоками.

Перші форми запису, що нагадують рахунки, з’явилися ще в Месопотамії близько 3200 року до н.е., де вони використовувались для ведення записів про сільськогосподарську продукцію. З того часу процес виписування рахунків значно еволюціонував.

У 1504 році відомий художник Ієронім Босх використав рахунок для вимоги платежу за свою картину. Він використав рахунок, щоб отримати оплату у розмірі 36 фунтів від короля Філіпа Красивого за свою картину 1504 «Страшний суд». На відміну від рахунків, які ми звикли бачити сьогодні, вони являли собою скоріше записку з розрахунками, подряпаними на полях (рис. 1.1) [4].

Перші рахунки, як правило, створювалися вручну, використовуючи папір та олівець або ручку. Продавці або керівники підприємств фіксували угоди та здійснені продажі вручну, зазначаючи в рахунках необхідну

інформацію, таку як назва товару чи послуги, ціна, кількість, загальна сума та інші важливі дані.

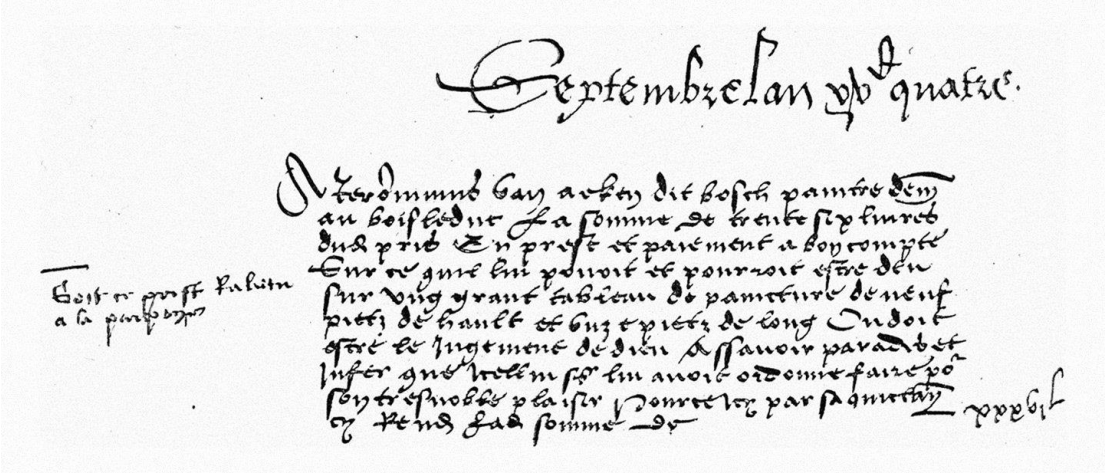


Рисунок 1.1 – Найстаріший рахунок у сучасній історії, написаний всесвітньо відомим художником Ієронімом Босхом

Ось декілька причин, чому рахунки стали невід'ємною частиною грошових відносин:

− облік фінансових операцій. Рахунки дозволяють фіксувати тазберігати інформацію про здійснені фінансові транзакції, такі як купівля-продаж товарів чи послуг, оплата рахунків, отримання доходів тощо. Цей облік є важливим для контролю над фінансами та аналізу ефективності бізнесу;

− доказ операцій. Рахунки є юридичними документами, що підтверджують здійснення фінансових операцій між сторонами. Вони служать доказом про здійснення певної угоди та вказують на умови операції, такі як ціна, кількість товарів чи послуг, умови оплати;

− податковий облік. Рахунки є важливим елементом для обліку податкових зобов'язань. Вони дозволяють підприємствам вести облік витрат та доходів, необхідних для розрахунку податків, таких як ПДВ, податок на прибуток, податок на додану вартість;

− аналіз фінансової діяльності. Рахунки забезпечують інформацію, необхідну для аналізу фінансової діяльності підприємства. З їхньою допомогою можна визначити обсяги продажів, витрати, прибуток, а також зробити прогнози та плани для майбутнього.

Таким чином, рахунки в грошових відносинах стали важливим інструментом для контролю, документації та аналізу фінансових операцій, що сприяло покращенню управління бізнесом та забезпечило більшу прозорість та відповідність у фінансових відносинах.

Сучасні комп'ютери та поява Інтернету змінили методи роботи багатьох підприємств, зокрема порядок виставлення рахунків. У 1960-х роках була розроблена перша EDI, яка дозволяла компаніям та їхнім торговим партнерам передавати такі документи, як рахунки-фактури та замовлення на покупку. Інформацію все ще потрібно було вводити вручну, перш ніж рахунки-фактури друкуватимуть і надсилатимуть поштою, але нова система скоротила час, необхідний для введення даних і більшості документів.

У 1990-х роках деякі компанії почали використовувати вебзастосунки для завантаження файлів EDI, що дозволяло постачальникам надсилати клієнтам рахунки на затвердження перед їхньою офіційною видачею. Це програмне забезпечення також можна використовувати для перегляду всієї історії виставлення рахунків та пошуку минулих рахунків, якщо це необхідно [4].

Автоматизація процесу створення рахунків почалася з розвитком комп'ютерних технологій та появою програмного забезпечення для бізнесу. Це дозволило автоматизувати процес формування рахунків, зменшивши час, необхідний для їхнього створення, а також усунуло ризик людських помилок. Автоматизація процесу створення рахунків значно покращила ефективність та точність фінансового обліку в підприємствах. Вона дозволила швидше та точніше виставляти рахунки клієнтам, що позитивно вплинуло на оборотність підприємств та сприяло підвищенню їхньої конкурентоспроможності.

Технології змінюють те, як ми взаємодіємо з усім, що нас оточує, і уряди

визнають важливість оцифрування в таких сферах, як виставлення рахунків. Інші країни ЄС уже показали, як бізнес стає ефективнішим, пропонує споживачам кращий досвід виставлення рахунків і зменшує ухилення від податків.

З 1 січня 2015 року в Україні реєстрація податкових накладних здійснюється лише в електронному вигляді. Тобто підприємства повинні реєструвати всі податкові накладні в Єдиному реєстрі податкових накладних (ЄРПН). Це пояснюється тим, що електронні рахунки-фактури можна легко відстежувати, вони менш схильні до людських помилок, зменшують ризик ухилення від сплати податків і, таким чином, зменшують суперечки або затримки платежів [5].

Тому компаніям потрібні доступні програми з потрібним функціоналом, які дозволять зручно вести фінансовий облік, автоматизуючи роботу своєї команди та контролюючи фінансові операції.

1.2 Переваги та недоліки використання вебзастосунків для створення рахунків

Вебзастосунок представляє собою програмне забезпечення, що функціонує на вебсервері, на відміну від традиційних десктопних програм, які запускаються на операційній системі пристрою користувача. Користувач може отримати доступ до програми через веббраузер.

Однією з ключових переваг онлайн-платформ для управління рахунками є економія часу та пам’яті пристрою. Користувачам не потрібно завантажувати та налаштовувати сторонню програму. Усе, зо потрібно для користування – це доступ в Інтернет, веббраузер та завантаження клієнтського інтерфейсу.

Ще однією перевагою є те, що для розповсюдження оновлень вебпрограми розробнику достатньо лише оновити програму на вебсервері, після чого остання версія стає доступною для всіх користувачів.

Використання вебсервера дозволяє уникнути зберігання даних клієнта на його локальній системі, забезпечуючи можливість доступу до них з будь-якого пристрою, на якому встановлений веббраузер. Це також допомагає уникнути втрати даних через несправність пристрою користувача. Більшість обчислень здійснюється на вебсервері, що значно зменшує загальне навантаження на систему користувача порівняно з десктопними застосунками.

Однак, слід зазначити наявність недоліку, а саме обмеження доступності вебзастосунків при відсутності з'єднання з Інтернетом та можливе значне уповільнення завантаження програми при низькій швидкості Інтернету у користувача.

На сьогоднішній день глобальний ринок неможливо уявити без сучасних технологій, що полегшують життя людству. Управління рахунками – це процес, який потребує часу та уваги, особливо для малих підприємств та самозайнятих осіб. Часто вони стикаються з проблемами ведення обліку своїх фінансів через відсутність спеціалізованого програмного забезпечення. Існуючі програмні рішення не завжди відповідають їхнім потребам або вимагають великих витрат на придбання та підтримку.

Розробка доступної системи управління рахунками може допомогти організаціям зменшити ймовірність людської помилки, збільшити швидкість обробки рахунків, зробити продуктивною та контролювати роботу співробітників, поліпшити відносини з клієнтами та зробити бізнес-процеси більш ефективними та прозорими [6-10].

1.3 Аналіз програм для ведення бухгалтерського обліку для бізнесу

Free Invoice – це безкоштовний вебзастосунок, призначений для онлайн-бухгалтерії з можливістю річної підписки. Перше, що бачить користувач, коли заходить на вебсайт – це головна сторінка, на якій є можливість створити накладну та рахунок (рис. 1.2).

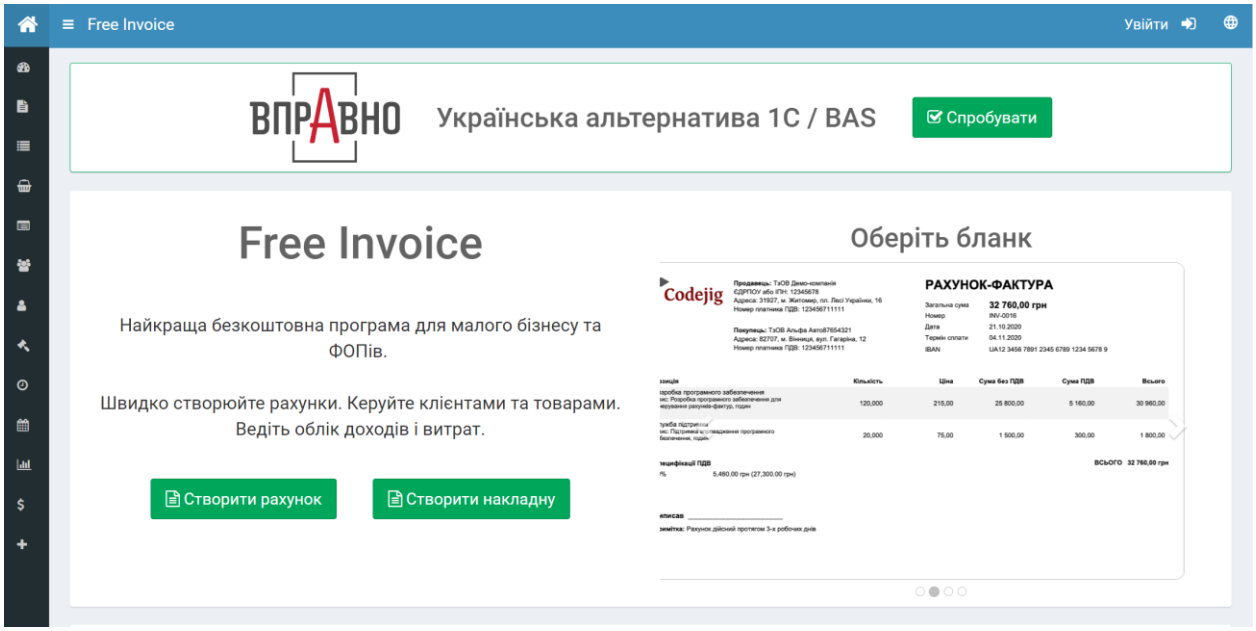


Рисунок 1.2 – Головна сторінка сайту Free Invoice

Ця програма пропонує непоганий спектр функціональних можливостей. У безкоштовній версії цієї програми можна зручно зберігати створені рахунки, базу клієнтів та товарів. Однак ця робота зосереджена на аспектах, пов’язаних із створенням рахунків-фактур вручну. Недоліком є те, що у вебзастосунку є тільки один обліковий запис для створення і користувачам-компаніям неможливо підключити декілька акаунтів для своїх працівників.

Нижче наведено наслідки відсутності цієї важливої функції:

– порушення безпеки даних компанії. Відсутність можливості реєстрації декількох облікових записів перешкоджає контролю доступу до конфіденційної інформації та порушує захист даних компанії;

– безвідповідальність у роботі бухгалтерського відділа компанії. Неможливо відстежувати, хто створює та змінює рахунки, що знижує прозорість та відповідальність;

– зменшує ефективність роботи працівників. Програма не дозволяє налаштувати рівні доступу, щоб оптимізувати робочий процес і зменшити ризик помилок.

Додатковим недоліком є необхідність ручного додавання нових товарів у безоплатній версії застосунку. Програма не дозволяє масово додавати список продуктів за допомогою файлів формату Excel. Користувач змушений вводити дані про товари вручну, що є трудомістким процесом (рис. 1.3).

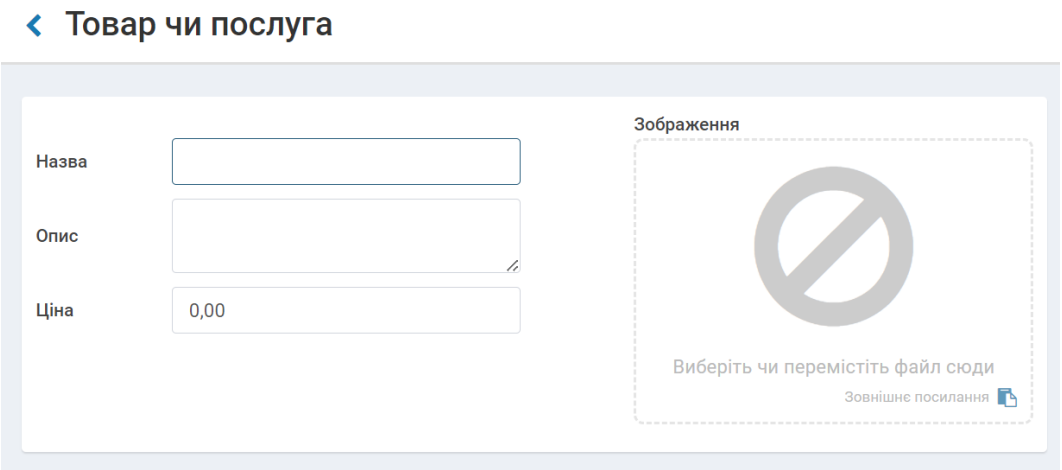


Рисунок 1.3 – Скріншот сторінки сайту Free Invoice

Іншим вебзастосунком в галузі управління рахунками є «Bitfaktura». Він представляє себе як онлайн-застосунок для створення рахунків, що характеризується інтуїтивно зрозумілим інтерфейсом користувача (рис. 1.4).

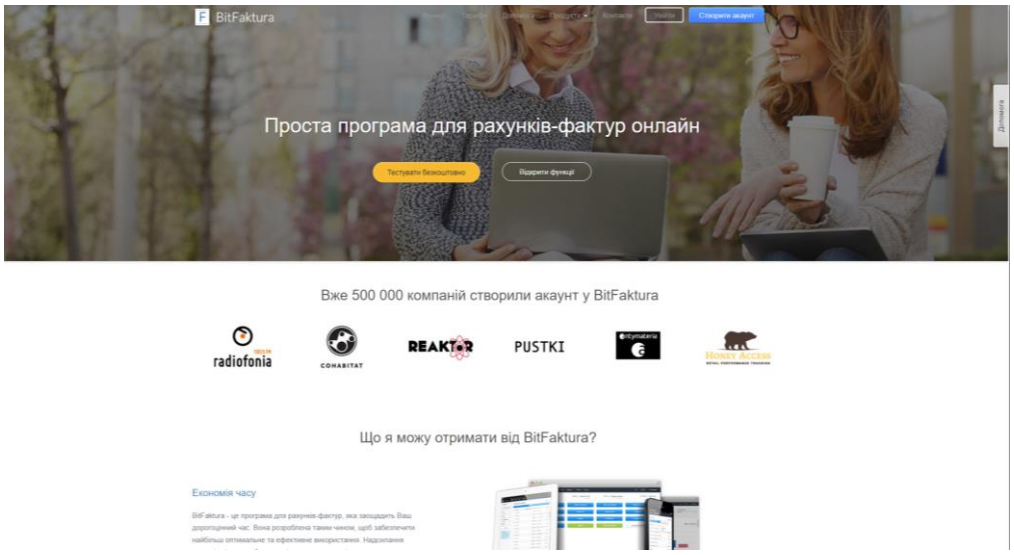


Рисунок 1.4 – Головна сторінка сайту Bitfaktura

Хоча програма пропонує детальні інструкції в базі знань на сайті для кожного розділу, що полегшує початок роботи, вона має незручне додавання співробітників власником або керівником компанії (рис. 1.5). Є необхідність ручного додавання нових співробітників керівником, яка породжує додаткове навантаження за часом.

Запропоноване рішення в рамках цієї кваліфікаційної роботи дозволяє працівнику самостійно реєструватися після отримання згоди керівника, що впливає на ефективність управління персоналом.

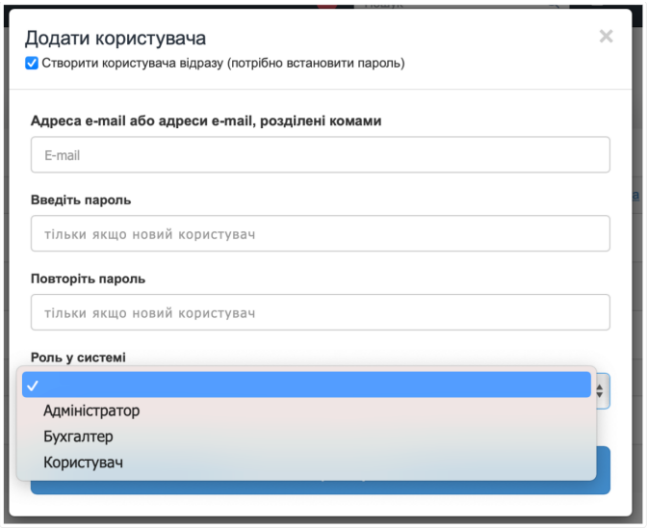


Рисунок 1.5 – Сторінка додавання вручну співробітника у програмі Bitfaktura

1.4 Постановка задачі

Актуальність розробки системи для управління рахунків визначається потребою бізнесу ефективно вести фінансовий облік.

Впровадження вебпрограми для управління рахунками з можливістю створення двох видів облікових записів: «Адміністратор» та «Співробітник», самостійною реєстрацією користувачів за попередніми запровенням та згодою електронною поштою, зручним додаванням даних про товари через файли Excel дозволить підприємцям зосередитися на розвитку своєї діяльності.

Система має автоматизувати роботу співробітників компанії та полегшити виписку рахунків для клієнтів компанії на продаж товарів, автоматично генеруючи рахунки та зменшуючи ризик людської помилки.

Об’єктом роботи є система управління рахунками.

Метою роботи є розробка системи управління рахунками для електронної комерційної діяльності бізнесу.

Для досягнення мети необхідно вирішити такі завдання:

- провести аналіз вебпрограм для управління рахунками, виявити їх переваги і недоліки;
- створити модель системи;
- розробити інтерфейс користувача;
- обрати архітектуру програми;
- програмно реалізувати систему управління рахунками.

2 АНАЛІЗ СПЕЦИФІКАЦІЇ ВИМОГ ТА СТВОРЕННЯ ІНТЕРФЕЙСУ СИСТЕМИ ДЛЯ УПРАВЛІННЯ РАХУНКАМИ

2.1 Обґрунтування вибору мов програмування

Усю роботу над проєктом в рамках кваліфікаційної роботи можна поділити на дві основні частини.

Перша – це бекенд розробка – створення серверної технічної частини проєкту, під якою розуміється все те, що не бачить звичайний користувач, наприклад: алгоритми, обробка, робота з базою даних і т.д. Бекенд обробляє та надає користувачеві дані, які потім відображає фронтенд.

Друга частина – це фронтенд розробка, та частина вебсайтів, яка доступна для користувача та з якою він може безпосередньо взаємодіяти. Вона охоплює відображення функціональних можливостей та користувацького інтерфейсу, які виконуються на боці клієнта, а також обробку запитів користувача. Загалом, фронтенд – це те, що спостерігає користувач під час відкриття вебсторінки.

Для створення структури всієї програми використовувалися такі мови програмування як: JavaScript, SQL, HTML і CSS.

Позначимо кожну мову окремо:

− мова програмування JavaScript, що використовується переважно для складних завдань у HTML-документах. Вона дозволяє змінювати інтерфейси вебзастосунків, додавати або змінювати різноманітні функціональні можливості сайту. Використання сторонніх API може дозволити додавати різноманітні функції, які вже існують на інших сайтах. Використання зовнішніх фреймворків і бібліотек в HTML-документі може прискорити процес створення вебсайтів і вебзастосунків;

− SQL використовується як ефективний спосіб для таких завдань, як збереження в базі даних, пошук їх частин, оновлення, вилучення з бази даних

та видалення звідти. Взаємодія з базою даних відбувається швидко, навіть у випадках, коли обсяги даних досить великі. SQL є досить зручною мовою для обробки великої кількості інформації, оскільки дозволяє розробнику миттєво перерозподіляти дані;

− мова програмування HTML, яка дозволяє розробнику створювати і структурувати різноманітні розділи, заголовки, посилання та блоки для вебсторінок і застосунків. HTML дозволяє створювати структуру вебсторінок;

− набір правил CSS для оформлення HTML-документів. Він дозволяє розробнику встановлювати макет сторінки, розмір шрифтів, кольори, заголовки, а також змінювати шрифт. Можна сказати, що CSS безпосередньо відповідає за візуальний вигляд сайту, оскільки жоден HTML-документ не може обійтися без CSS [11-14].

2.2 Специфікація вимог до функціоналу програми

Для того, щоб повністю представити функціонал вебзастосунку, було розроблено ряд функціональних та нефункціональних вимог, які є важливою частиною процесу розробки програмного продукту.

Функціональні вимоги описують, що повинна робити система. Вони визначають функції та можливості продукту, які виявляє користувач. Наприклад, це може бути функція входу в систему або розрахунку податку з продажу. Функціональні вимоги складаються з двох частин: функції (що виконує система) та поведінки (як система це робить).

Нефункціональні вимоги накладають обмеження на систему. Вони визначають атрибути якості системи, такі як продуктивність, безпека, надійність, тощо. Ці вимоги не впливають на функціональність програми, але важливі для забезпечення якості продукту [15,16].

Вимоги створено за допомогою методики SMART, що описує основні характеристики ефективних цілей та розшифровується, як Specific (конкретні),

Measurable (вимірні), Achievable (досяжні), Relevant (релевантні) и Time-bound (обмежені у часі) [17].

У структурі системи можна виділити два типи користувачів: адміністратор і співробітник.

Адміністратор – це будь-хто, хто зареєстрував свою компанію в програмі, користувач з найширшим спектром прав у системі. Має можливість керувати налаштуваннями та додавати нових співробітників. Адміністратора від співробітника відрізняє можливість переглядати всі рахунки, створені всіма користувачами, управляти співробітниками та списками товарів.

Співробітник працює безпосередньо з програмою та використовує основні функції програми, а саме створення рахунків.

На рисунку 2.1 наведено контекстну діаграму користувачів системи.

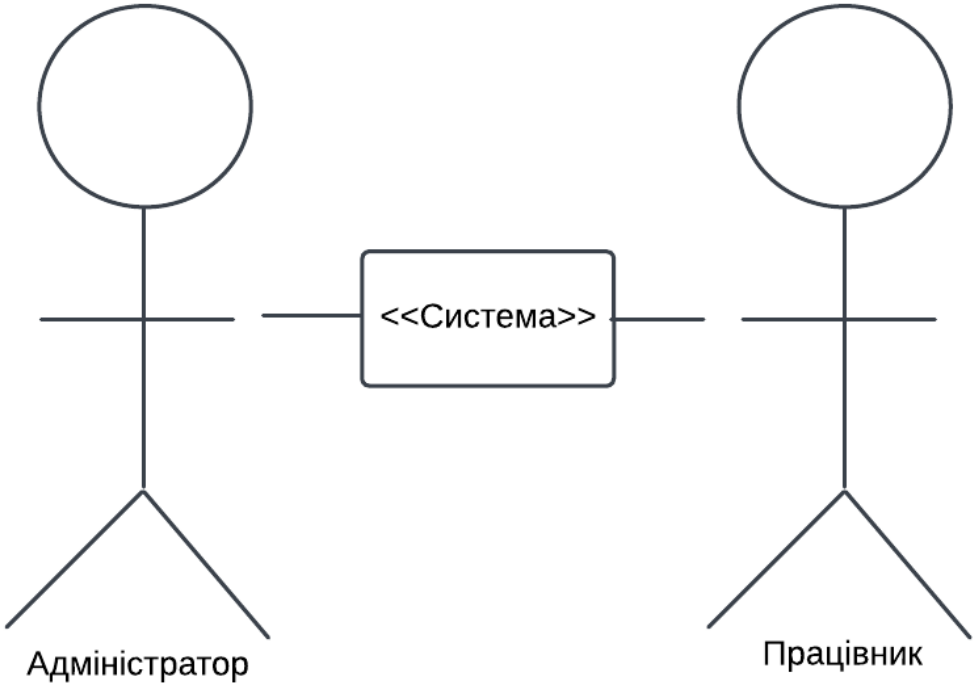


Рисунок 2.1 – Контекстна діаграма 0-рівня

Щоб створити рахунок, користувач вибирає продукт із доступного списку, вказує такі деталі, як відсоток ПДВ і відсоток знижки, а потім вводить

дані клієнта, а ідентифікаційні дані система вводить автоматично, беручи їх з облікового запису зареєстрованої компанії та бази даних клієнтів.

Отже, сформуємо функціональні вимоги до програми для створення рахунків:

– застосунок зберігає інформацію про користувачів, їхні авторизації, створені рахунки, інформацію про клієнтів і компанію;

– адміністратор зобов'язаний зберегти унікальну адресу електронної пошти, ім'я користувача та назву компанії у профілі;

– користувач зобов'язаний зберегти унікальну адресу електронної пошти у профілі;

– тільки одна особа може використовувати один обліковий запис користувача;

– адміністратор може використовувати всі функціональні можливості, доступні користувачеві;

– адміністратор може додати клієнта. Ім'я клієнта та ідентифікаційний податковий номер мають бути унікальними;

– адміністратор може створювати кілька категорій. Категорія може мати підкатегорії. Товари можна додати до категорії, ввівши кожен товар окремо. Також можна розмістити файл Excel зі списком товарів даної категорії. Після прочитання файлу система показує стовпці, які вона зрозуміла, просить користувача підтвердити назви стовпців і вказати тип даних у стовпці;

– система повинна розраховувати ПДВ, обчислюючи податок за окремими позиціями в рахунку та додаючи його;

– для кожної позиції в рахунку-фактурі вказуються знижки, вартість брутто та нетто товару;

– система повинна пам'ятати відомості про всіх доданих клієнтів, дані компанії, усі рахунки та їх реквізити, усі пропозиції;

– система повинна мати вебінтерфейс;

– система має дозволяти фільтрувати та сортувати рахунки за ціною та датою та користувачем;

– співробітник може переглядати рахунки, створені ним. Однак лише адміністратор може переглядати всі рахунки всіх користувачів;

– система повинна дозволяти перегляд рахунку, що створюється;

– рахунок можна завантажити у форматі pdf;

– система має можливість надсилати користувачам запрошення на реєстрацію електронною поштою з текстом, наданим раніше в листі;

– система має можливість прийняти нового користувача в якості адміністратора.

Сформуємо нефункціональні вимоги до системи для створення рахунків:

– основні мови, які має підтримувати програмне забезпечення: англійська;

– система має бути сумісною із затвердженим проєктом;

– інтерфейс програми має бути зрозумілим. Оскільки користувальницький інтерфейс системи використовується ефективно, очікується, що користувач без попередньої підготовки зможе завершити основний варіант використання менш ніж за кілька годин;

– запуск програми в різних середовищах. Програмне забезпечення має підтримувати принаймні такі браузери: Google Chrome 104, Safari 12.1.2, Edge 88.

На основі функціональних та нефункціональних вимог до проєкту було побудовано діаграму взаємозв’язку сутності вебзастосунку. Це візуальна модель, яка показує зв'язок сутностей у базі даних. У якому сутність цієї бази даних посилається на об'єкти або компоненти, які визначають властивості компанії. Діаграма зображена на рисунку 2.2 [18].

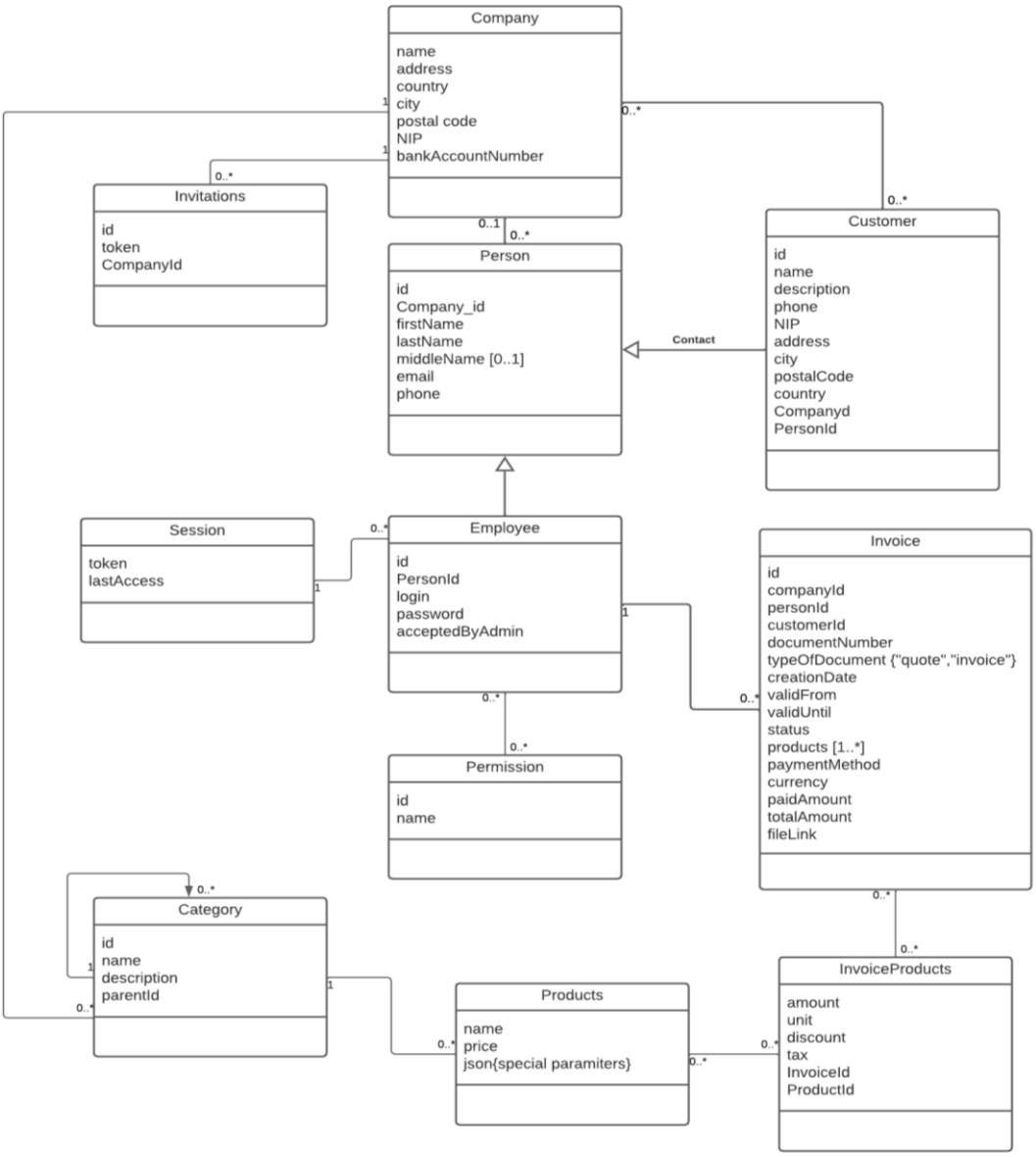


Рисунок 2.2 – Діаграма взаємозв’язку сутності

2.3 Варіанти та сценарії використання

Основний функціонал програми полягає у створенні замовлень користувачем і додавання товарів до бази товарів. Нижче на рисунках 2.3, 2.4 та 2.5 наведено випадки використання користувачів з дозволом «Адміністратор» та «Співробітник». Діаграми варіантів використання були створені за допомогою Lucidchart [19].

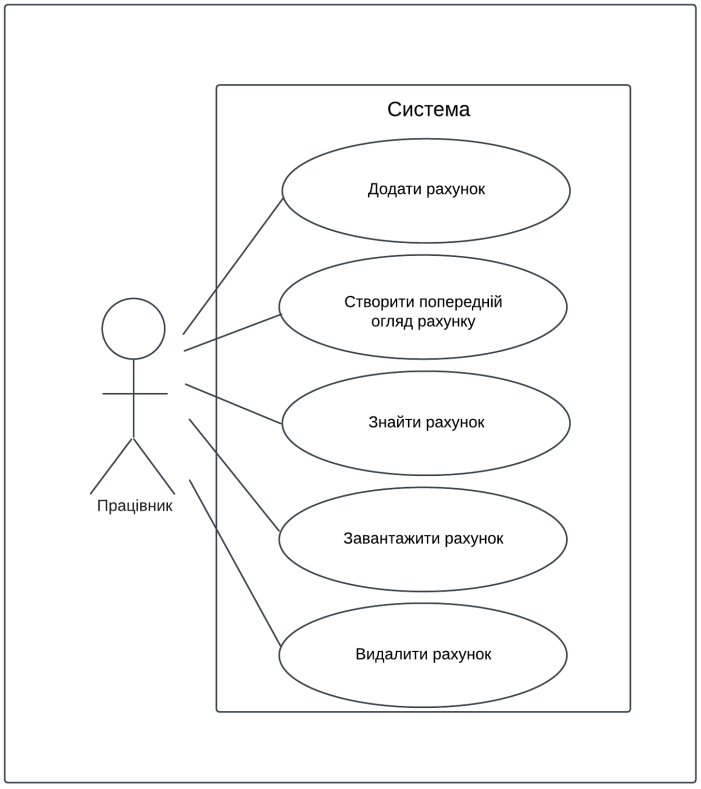


Рисунок 2.3 – Діаграма варіантів використання, що представляє дії співробітника під час додавання рахунку-фактури.

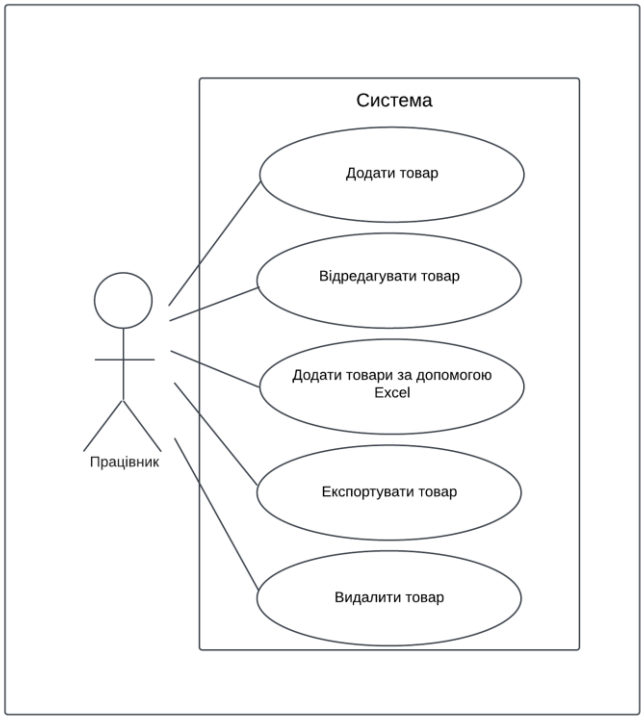


Рисунок 2.4 – Діаграма діяльності співробітника в контексті додавання продукту

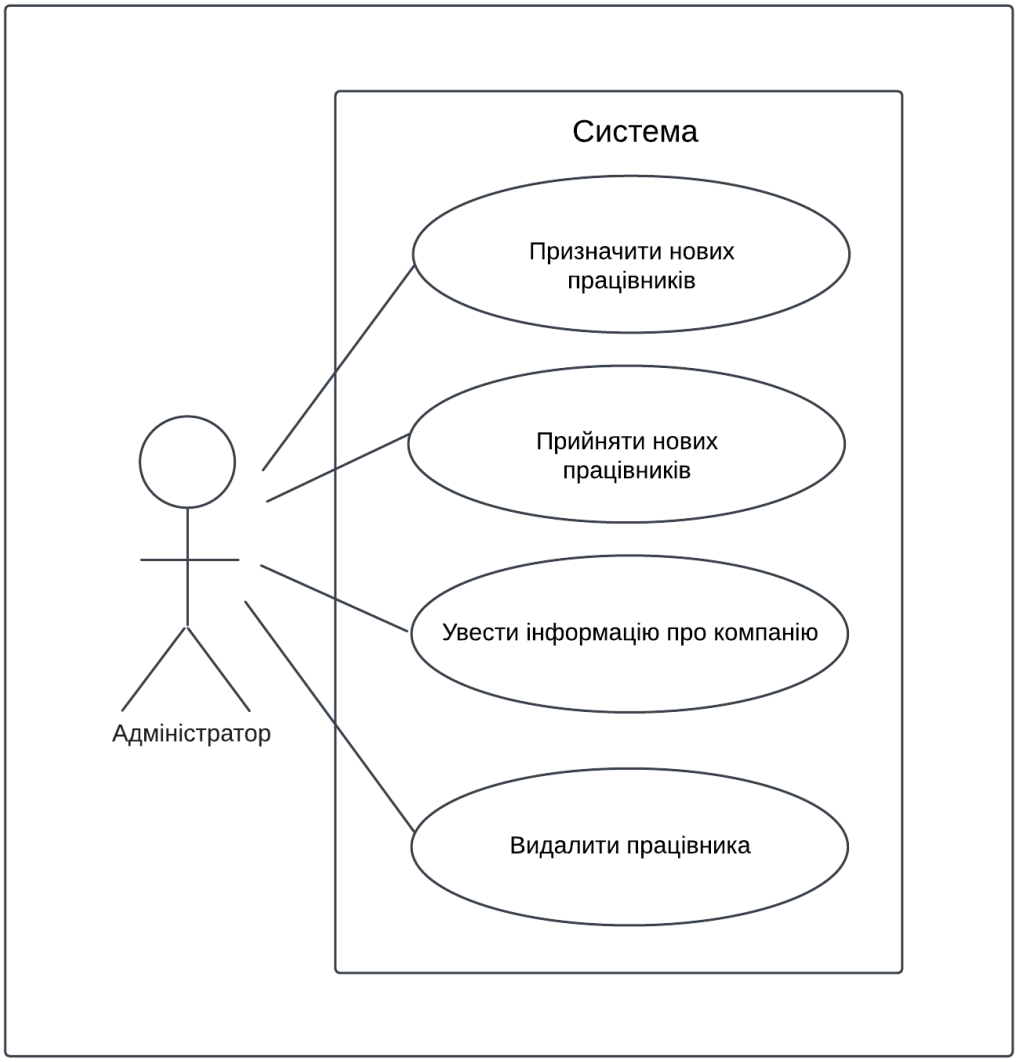


Рисунок 2.5 – Діаграма варіантів використання для адміністратора.

Сценарій використання «Додайте співробітника» використовується для реєстрації співробітника в застосунку. Цей варіант використання призначений для актора «Адміністратор». Нижче наведено таблицю з основним сценарієм (табл. 2.1).

Варіант використання «Додайте категорії» використовується для реєстрації категорій у програмі. Цей варіант використання для актора "Співробітник". Нижче наведено таблицю 2.2 зі сценарієм, що охоплює основний та альтернативний потік подій.

Таблиця 2.1 – Сценарій для випадку «Додати співробітника»

Етап	**Дія**
Випадок використання	Додайте співробітника
Початковий стан	Відсутній
Потік основних подій	1. Користувач (актор) запускає варіант використання. 2. Актор натискає кнопку «Налаштування». 3. Система відображає налаштування застосунку. 4. Актор вибирає вкладку «Облікові записи». 5. Система представляє налаштування облікового запису користувача. 6. Актор вводить електронну адресу нового співробітника. 7. Система повідомляє: «Посилання на реєстрацію надіслано». 8. Адміністратор отримує електронний лист із запитом на згоду та посилання для підтвердження. 9. Актор відкриває посилання з електронного листа. 10. Система направляє актора на вкладку налаштувань. 11. Актор знову вибирає вкладку «Облікові записи». 12. Система представляє налаштування облікового запису користувача. 13. Актор підтверджує додавання нового співробітника, натиснувши «Прийняти». 14. Новий співробітник отримує електронний лист із підтвердженням реєстрації. 15. Користувач завершує дію.
Альтернативні потоки подій	6а. Актор натискає кнопку «копіювати». 6а1. Система відображає «скопійовано». 13а. Актор натискає «видалити». 13а1. Користувач завершує дію.
Кінець	У будь-який час
Кінцевий стан	У базі вебзастосунку зареєстровано нового співробітника.

Таблиця 2.2 – Сценарій для випадку «Додати категорію»

Етап	**Дія**
Випадок використання	Додайте категорію
Початковий стан	Відсутній
Потік основних подій	1. Користувач (актор) запускає варіант використання. 2. Актор натискає кнопку «+» біля слова «Категорія». 3. Система відобразить сторінку «Додати категорії». 4. Актор обирає створення категорії без підкатегорій. 5. Актор вводить назву категорії. 6. Актор натискає кнопку «Надіслати». 7. Система завершує роботу.
Альтернативні потоки подій	4а. Актор вибирає параметри «Підкатегорії». 4а1. Актор вводить назву підкатегорії.
Кінець	У будь-який час
Кінцевий стан	У базі даних програми зареєстровано нові категорії. Альтернативно: нові категорії з підкатегоріями будуть зареєстровані в базі даних вебзастосунку.

Випадок використання «Додати продукт вручну» використовується для реєстрації категорій у програмі. Цей варіант використання для актора «Співробітник». Нижче наведено таблицю 2.3 зі сценарієм, що охоплює основний та альтернативний потік подій.

Випадок використання «Додати продукти з файлу Excel» використовується для реєстрації категорій у програмі. Цей варіант використання для актора «Співробітник». У таблиці 2.4 розписаний сценарій, що охоплює основний та альтернативний потоки подій. Додатково була представлена діаграма діяльності (рис. 2.6)

Таблиця 2.3 – Сценарій для випадку «Додайте продукт вручну»

Етап	**Дія**
Випадок використання	Додайте продукт вручну
Початковий стан	Відсутній
Потік основних подій	1. Користувач (актор) запускає варіант використання. 2. Актор натискає на раніше створену категорію або підкатегорію. 3. Система відобразить сторінку «Продукти». 4. Актор натискає «Додати новий продукт». 5. Актор вводить назву стовпця, дані та тип даних. 6. Актор натискає кнопку «Надіслати». 7. Система завершує роботу.
Кінець	У будь-який час
Кінцевий стан	До бази даних додано новий продукт.

Таблиця 2.4 – Сценарій для випадку «Додати продукти з файлу Excel»

Етап	**Дія**
1	**2**
Випадок використання	Додайте продукти з файлу Excel
Початковий стан	Відсутній
Потік основних подій	1. Працівник (актор) ініціює варіант використання. 2. Актор вибирає наявну категорію або підкатегорію. 3. Система представляє сторінку «Продукти». 4. Актор вибирає «Імпортувати з файлу». 5. Система представлена інтерфейсом «перетягування». 6. Актор завантажує файл Excel. 7. Система запитує номер рядка з назвами стовпців. 8. Актор вводить потрібний номер рядка. 9. Система представляє «Налаштувати відображення стовпців» із експортованими стовпцями. 10. Актор налаштовує стовпці: призначає типи даних і вирішує, які з них відображатимуться в рахунку-фактурі. 11. Актор підтверджує внесені зміни. 12. Система переносить актора до огляду продукту. 13. Процес завершено.

1	2
Альтернативні потоки подій	6a. У файлі Excel виявлено кілька аркушів. 6a1. Система пропонує на вибір: «Обробити лише вибраний аркуш» або «Створити нову категорію для кожного аркуша». 6a1.a. Актор вибирає «Обробити лише вибраний аркуш» і вказує відповідний аркуш і номер рядка з назвами стовпців. 6a1.b. Актор вибирає «Створити нову категорію для кожного аркуша», вказує номер рядка з назвами для кожного аркуша, потім налаштовує стовпці та підтверджує зміни. 6a1.b3. Система покаже сторінку «Нова категорія з аркуша». 6a1.b4. Актор редагує назви категорій. 6a1.b5. Актор редагує стовпці, призначає типи даних у стовпці та вибирає, чи показувати його в рахунку-фактурі. 6a1.b6. Актор натискає «Надіслати». 6a1.b7. Система перенаправляє актора на сторінку товару. 6a1.b8. Процес завершується.
Кінець	У будь-який час
Кінцевий стан	До бази даних програми додано новий продукт.

Випадок використання «додати рахунок-фактуру» використовується для реєстрації категорій у програмі.

Цей варіант використання для актора «Співробітник». Нижче наведено таблицю 2.5 зі сценарієм, що охоплює основний та альтернативний потік подій.

Таблиця 2.5 – Сценарій для випадку «Додайте рахунок-фактуру»

Етап	Дія
1	2
Випадок використання	Додайте рахунок-фактуру

1	2
Початковий стан	Відсутній
Потік основних подій	1. Користувач (актор) запускає варіант використання. 2. Система виводить форму для додавання рахунку. 3. Актор вибирає клієнта з наявної клієнтської бази. 4. Актор додає дату закінчення рахунку-фактури. 5. Актор вибирає продукт зі списку товарів. 6. Актор обирає валюту та спосіб оплати. 7. Актор додає дату оплати. 8. Актор натискає «Створити», щоб зберегти рахунок-фактуру в базі даних програми
Кінець	У будь-який час
Кінцевий стан	У застосунку (базі даних) зареєстровано новий рахунок.

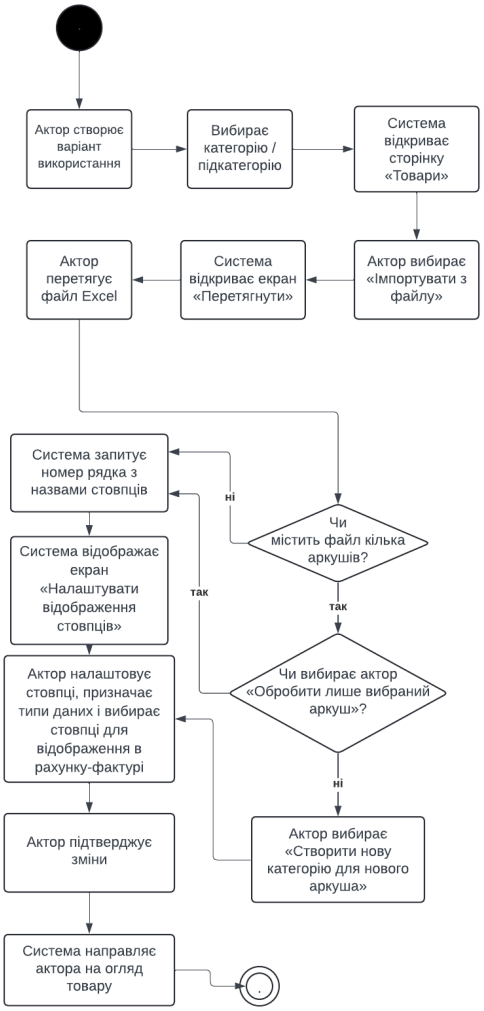


Рисунок 2.6 – Діаграма діяльності для випадку «Додати продукти з файлу Excel»

2.4 Розробка інтерфесу вебзастосунка

2.4.1 Розробка прототипів

Перше, що було зроблено для візуального представлення інтерфейсу вебресурсу для створення рахунків – це створення прототипів.

Створення прототипів є важливим кроком у розробці продукту, тому що це:

– допомагає переконатися, що створюється правильний функціонал програми;

– допомагає краще зрозуміти майбутній продукт;

– дозволяє перевіряти ідеї та гіпотези набагато швидше і з меншим ризиком [16].

Прототипи було розроблено за допомогою онлайн-сервіса для проєктування інтерфейсу та створення прототипів Figma. Він дає можливість розробити UI/UX онлайн-дизайн, працювати в реальному часі та створювати презентацію прототипу [20-23].

Нижче описані основні проблеми пов’язані зі створенням прототипів та їх вирішення. Мета полягала у розробці інтуїтивно зрозумілого візуального представлення рахунків. Рішенням цієї проблеми є дизайн сторінки «Створення рахунку», створений за стандартами оформлення рахунку як документу (рис. 2.7).

Наступною задачею стало представлення даних усієї категорії товарів. Відображення списку продуктів було складним завданням для забезпечення ефективності та естетики, щоб полегшити користувачам інтуїтивно зрозумілий досвід під час перегляду та пошуку конкретних продуктів. Ключовим питанням було уникнути перевантаження інформацією та потенційної плутанини користувачів. Рішення цієї проблеми представлено на рисунку 2.8.

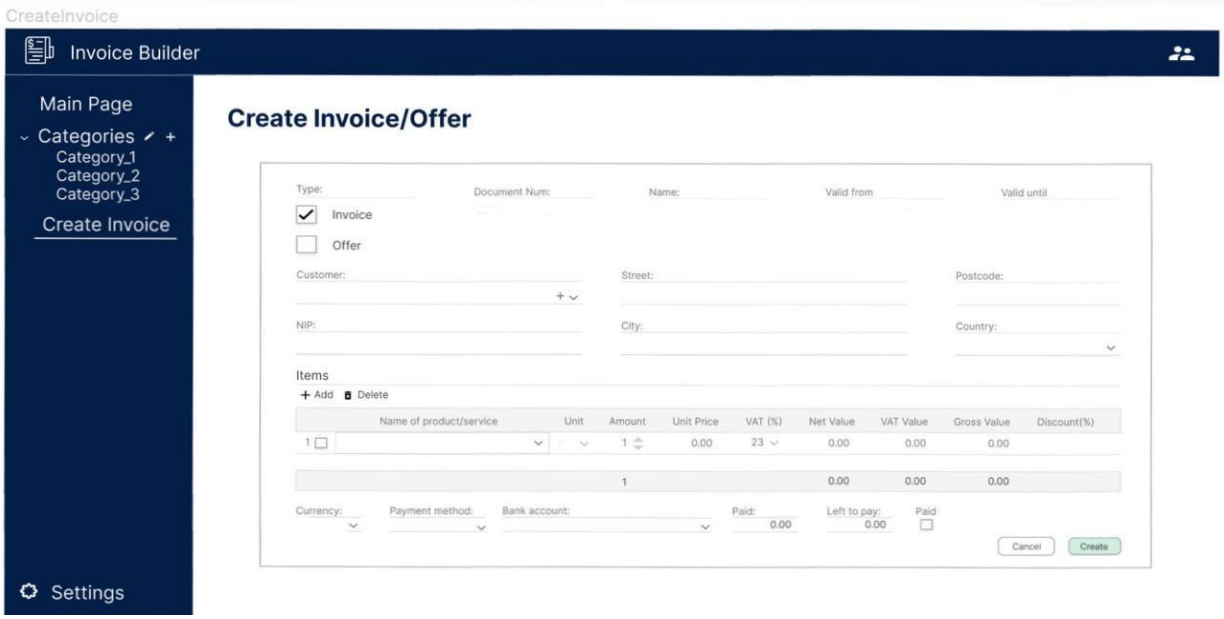


Рисунок 2.7 – Прототип сценарію використання «Додати рахунок-фактуру»

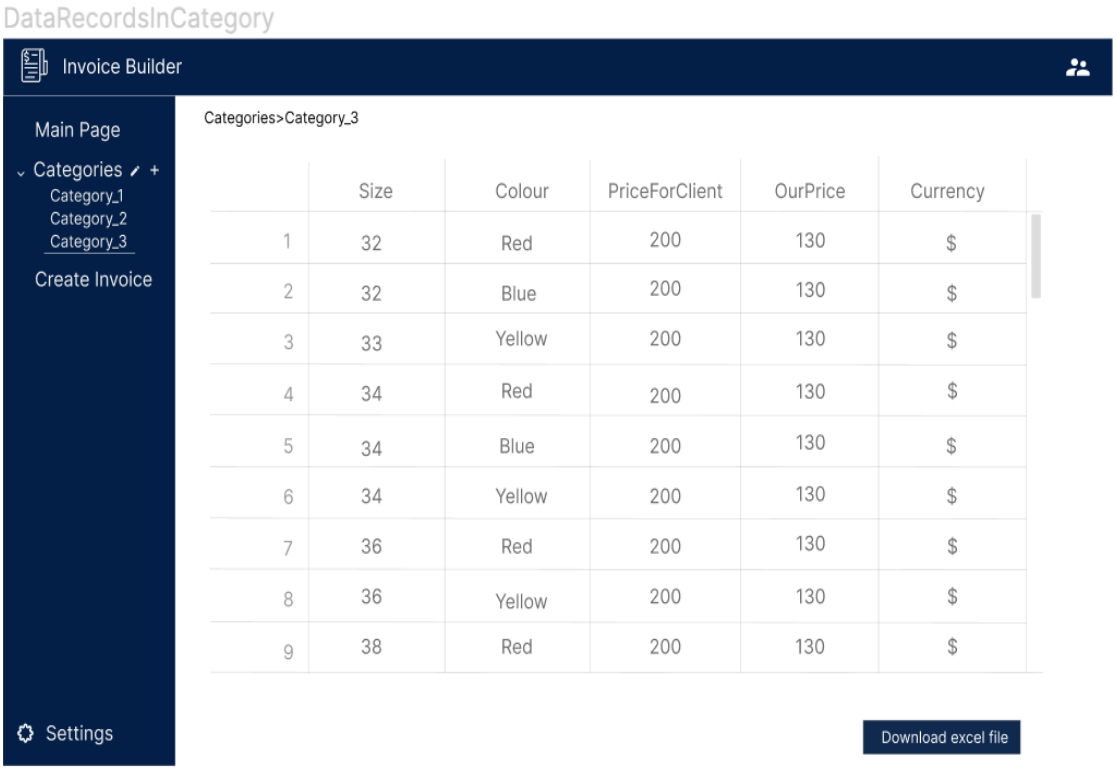


Рисунок 2.8 – Прототип сторінки «Продукти».

В рамках кваліфікаційної роботи у процесі створення застосунку була використана функція, яка після надсилання та прийняття Excel файлу автоматично представляє користувачеві виявлені стовпці даних. Далі система

попросить користувача перевірити та підтвердити тип кожного з цих даних. Рішення цієї задачі представлено на рисунку 2.9.

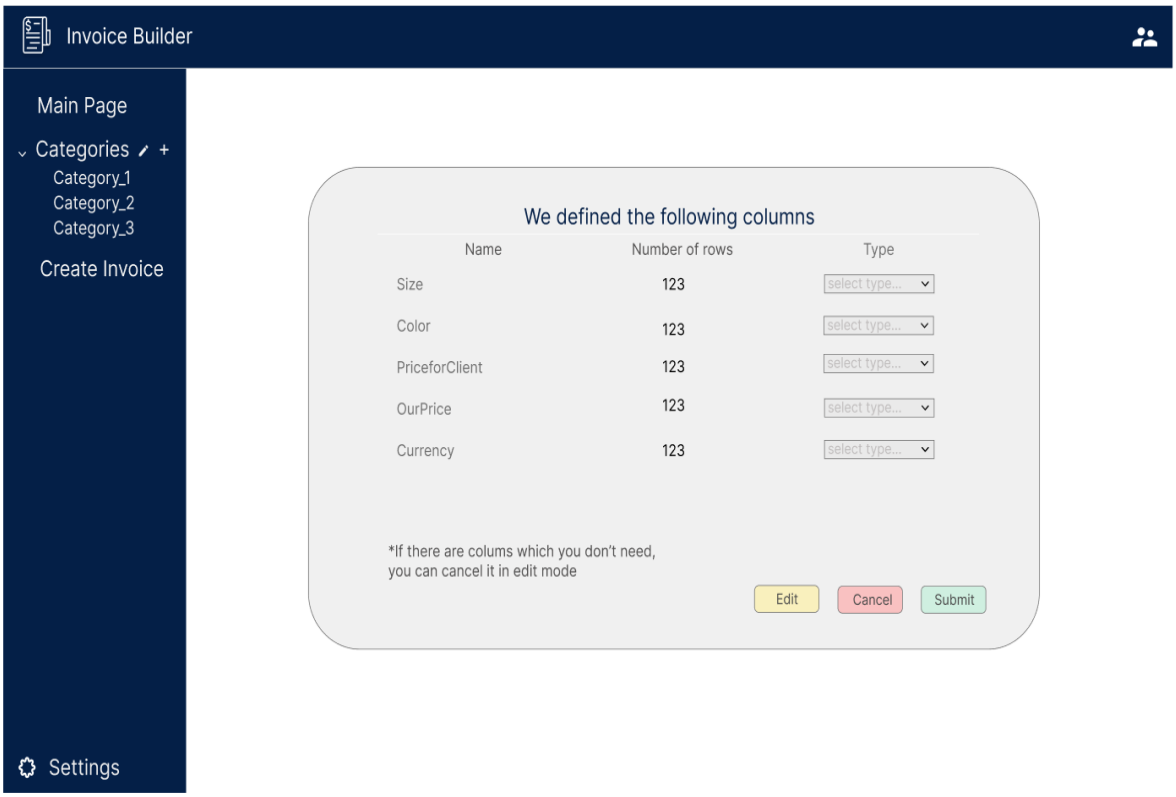


Рисунок 2.9 – Прототип сторінки «Налаштувати відображення стовпців».

2.4.2 Опис інтерфейсу програми

Мова інтерфейсу за замовченням програми була обрана англійська, адже враховано, що це дає можливість українському вебзастосунку за походженням вийти на міжнародний рівень користування у майбутньому, а користувачам виставляти рахунки не тільки у межах України, а й за кордон.

Перше, що бачить користувач, коли заходить на вебсайт – це сторінка входу. Щоб компанії-користувачу почати користування застосунком, треба зареєструватися (рис. 2.10).

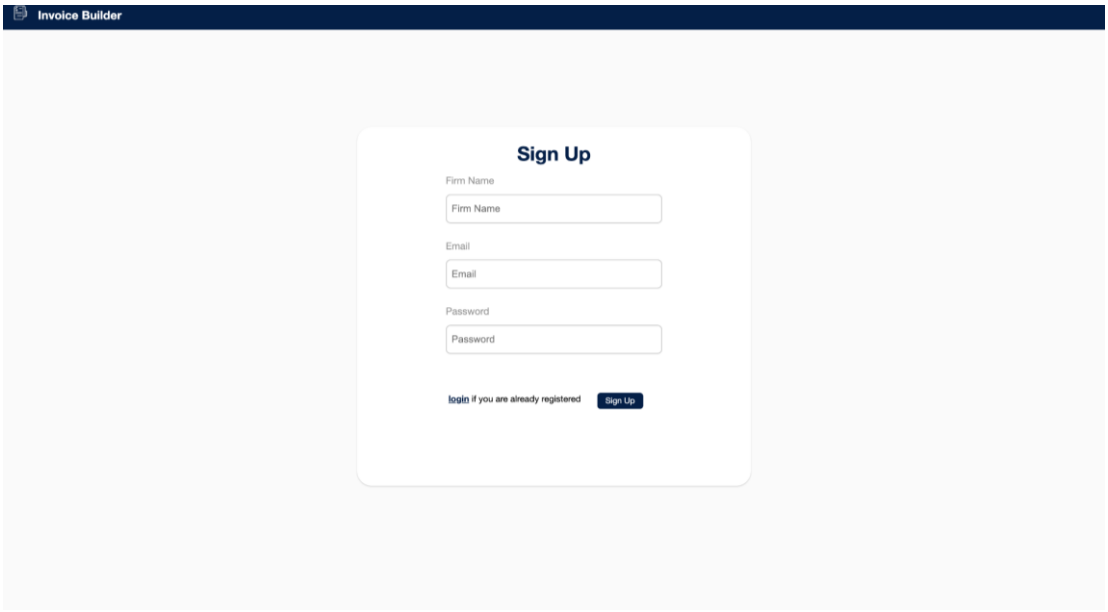


Рисунок 2.10 – Сторінка реєстрації користувача

Якщо у користувача вже є обліковий запис, він може перейти на сторінку входу, вибравши «Увійти, якщо ви вже зареєстровані» (рис. 2.11).

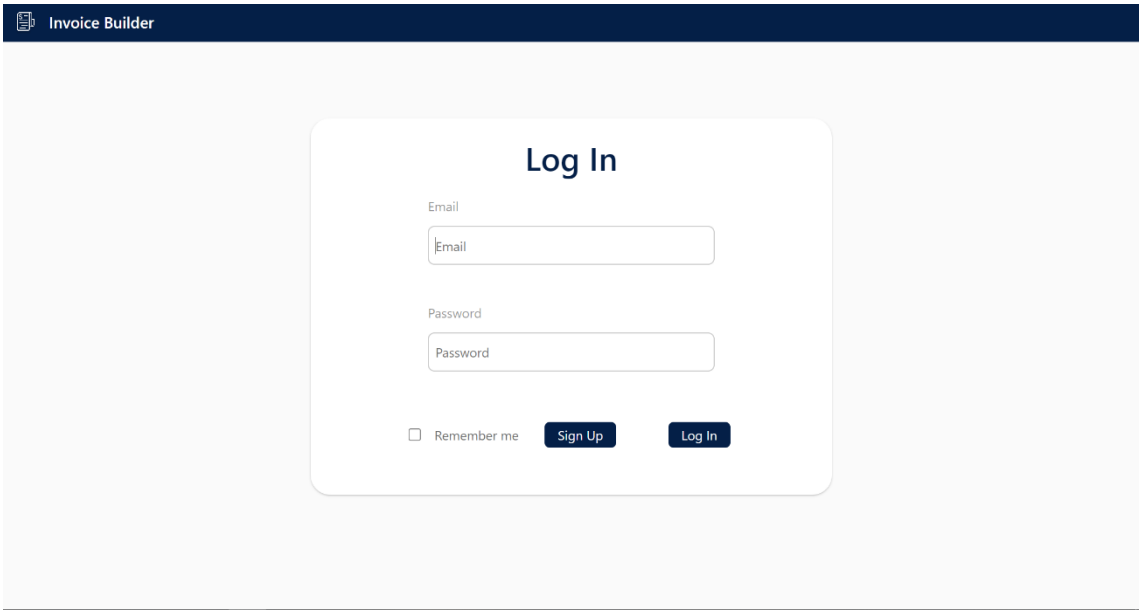


Рисунок 2.11 – Сторінка входу для користувачів

Після входу користувач переходить на головну вкладку «Рахунки» (рис. 2.12). Тут представлені всі створені рахунки. Користувачі з дозволом «Адміністратор» мають доступ до перегляду рахунків-фактур, створених

усіма користувачами. Користувачі з дозволом «Співробітник» можуть переглядати лише власні рахунки.

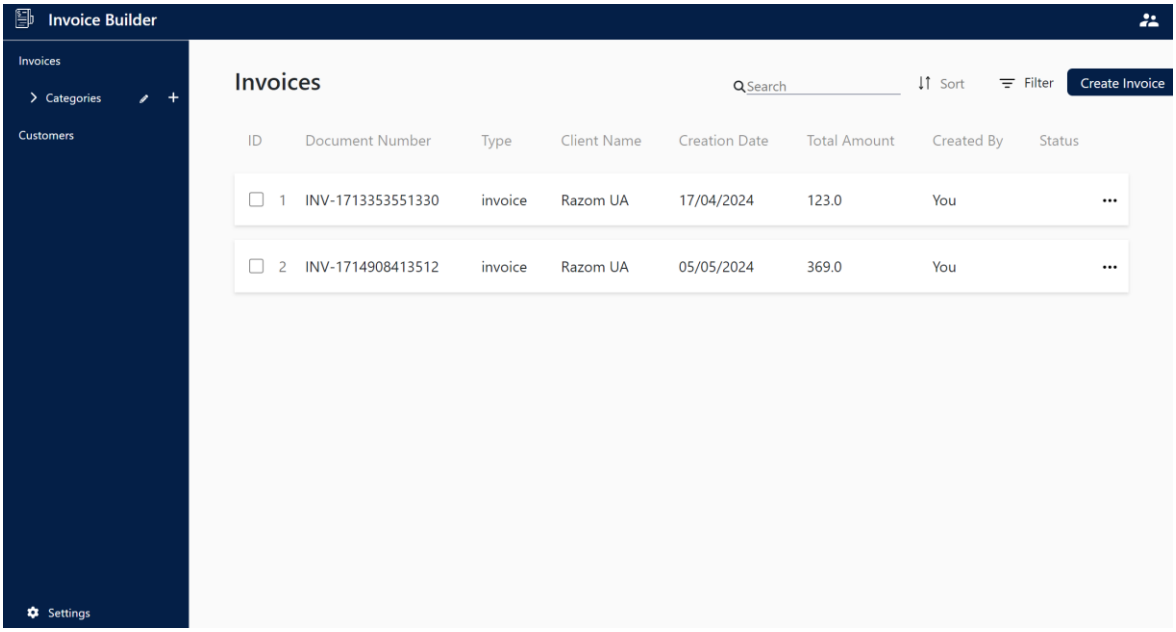


Рисунок 2.12 – Головна сторінка

Після реєстрації користувачу важливо заповнити дані компанії у вкладці «Налаштування». Це має право робити лише користувач із правами адміністратора. Без заповненої інформації про компанію неможливо буде користуватися основним функціоналом програми, яким є формування рахунків, адже основною перевагою застосунку є те, що дані про компанію автоматично заповнюються у згенерованому документі рахунку. Це можна зробити на наступному екрані, що представлений на рисунку 2.13.

Також важливо заповнити дані у вкладці «Облікові дані», які будуть відображатися у створеному документі (рис. 2.14).

Вкладка «Облікові записи» надає функціонал, що дозволяє переглядати список співробітників компанії (рис. 2.15). Крім того, є можливість запрошувати нових користувачів за спеціальним посиланням або функцією «Надіслати посилання електронною поштою», яка дозволяє надіслати електронного листа із запрошенням потенційному співробітнику. Після

завершення процесу реєстрації новий користувач не може увійти, поки адміністратор не підтвердить це в панелі «Облікові записи».

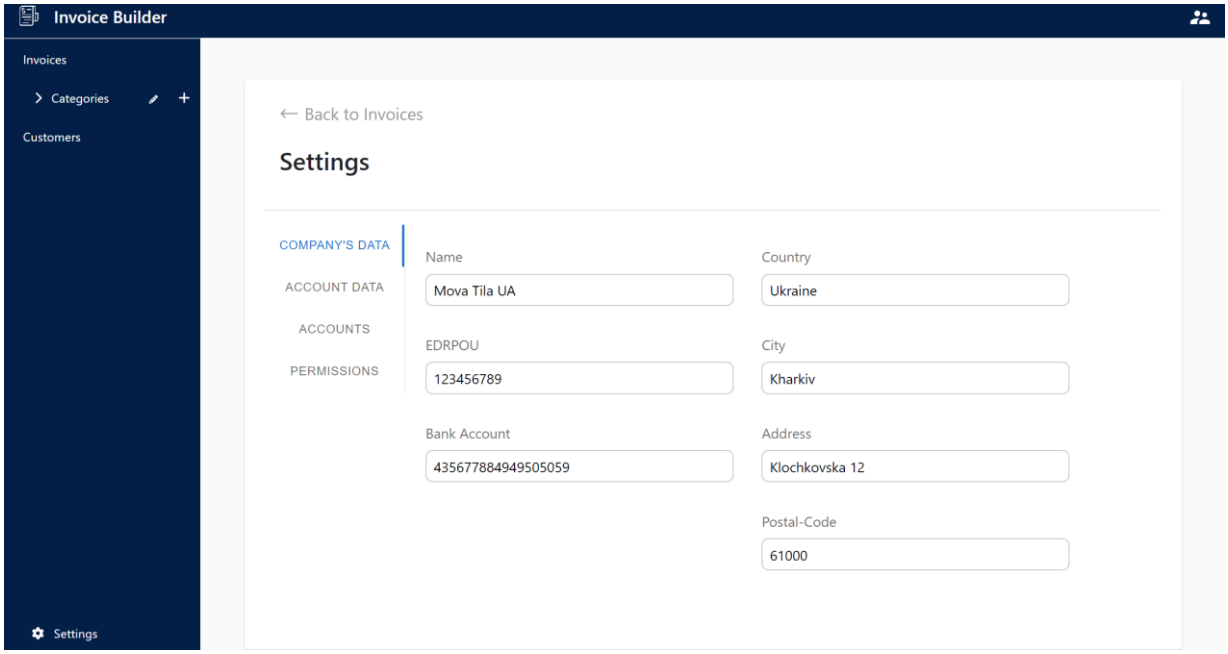


Рисунок 2.13 – Приклад заповнення даних про компанію-користувача

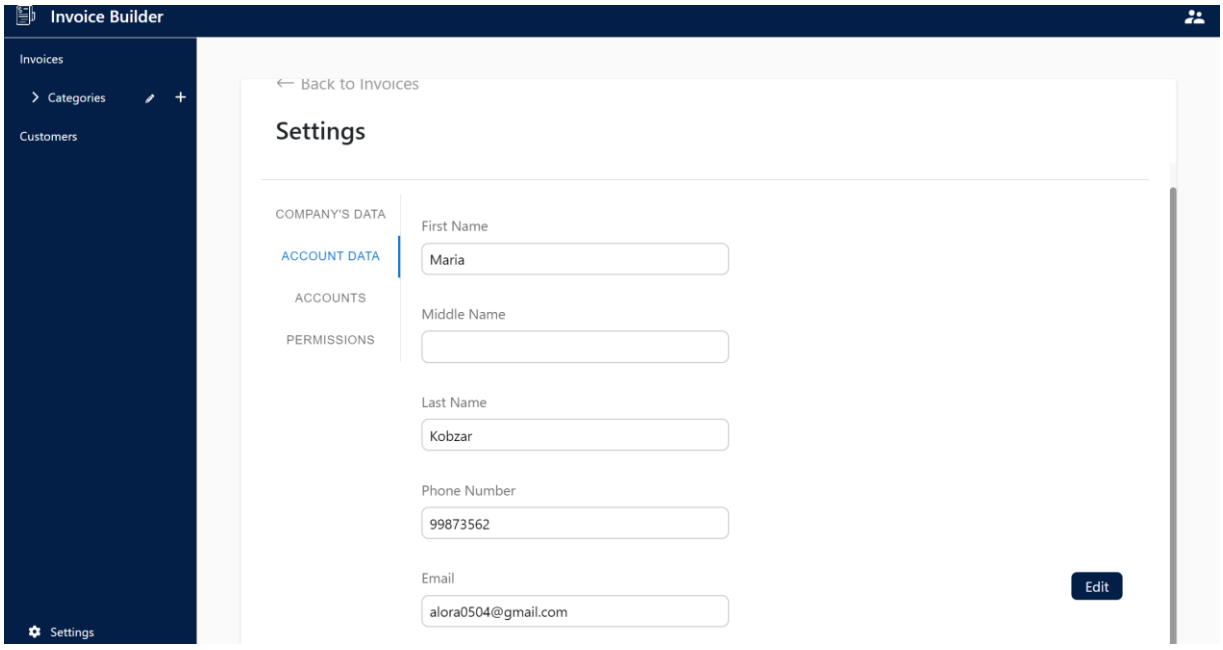


Рисунок 2.14 – Екран налаштувань даних користувача

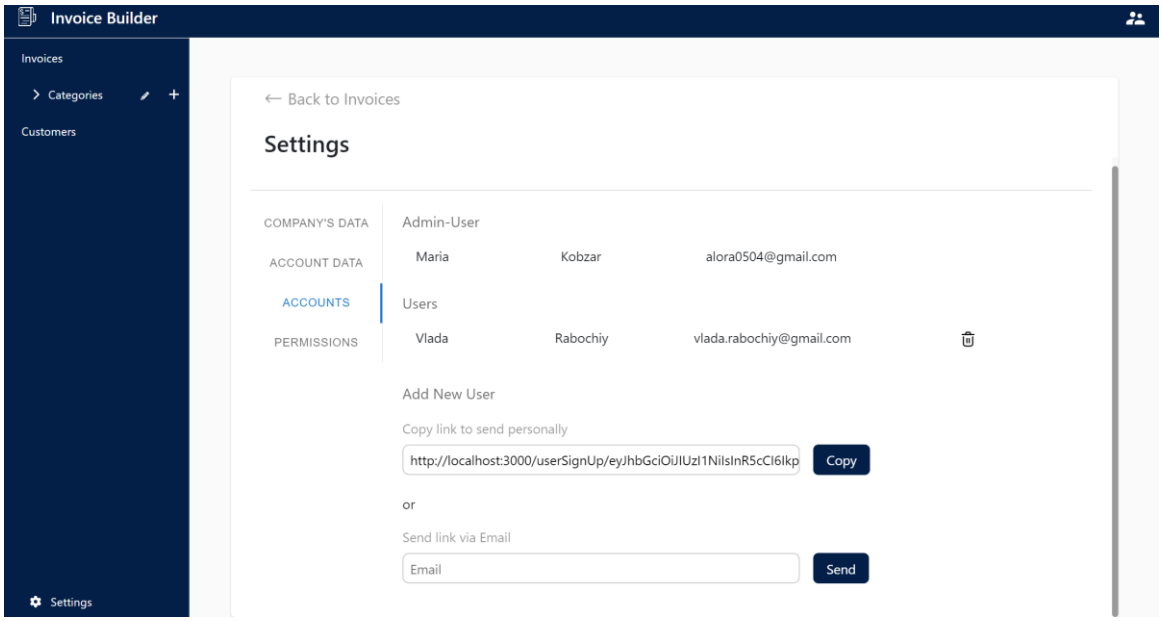


Рисунок 2.15 – Екран налаштувань облікового запису компанії

Після успішної реєстрації та схвалення адміністратором користувач отримує електронний лист із підтвердженням активації облікового запису (рис. 2.16).

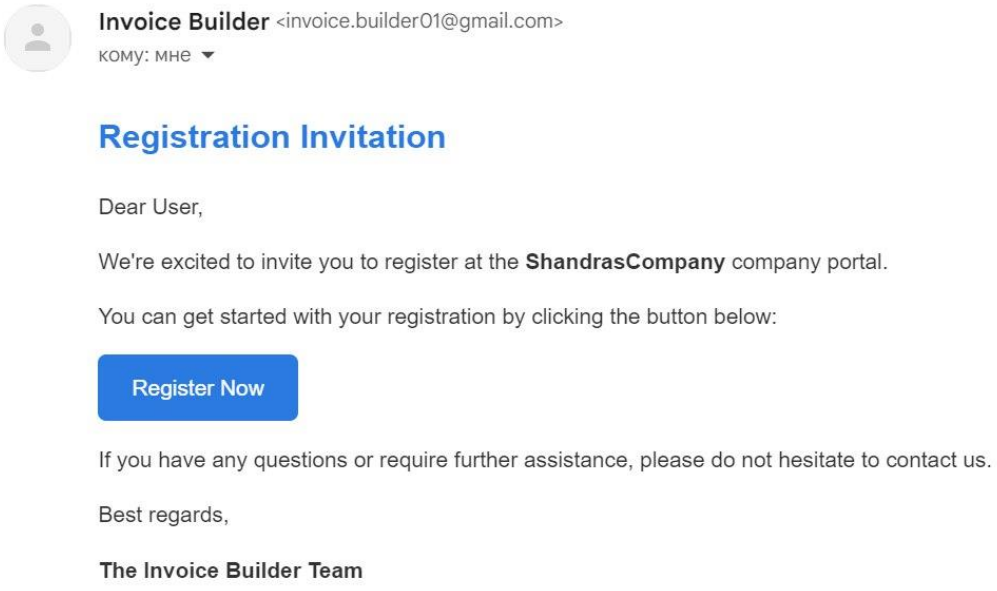
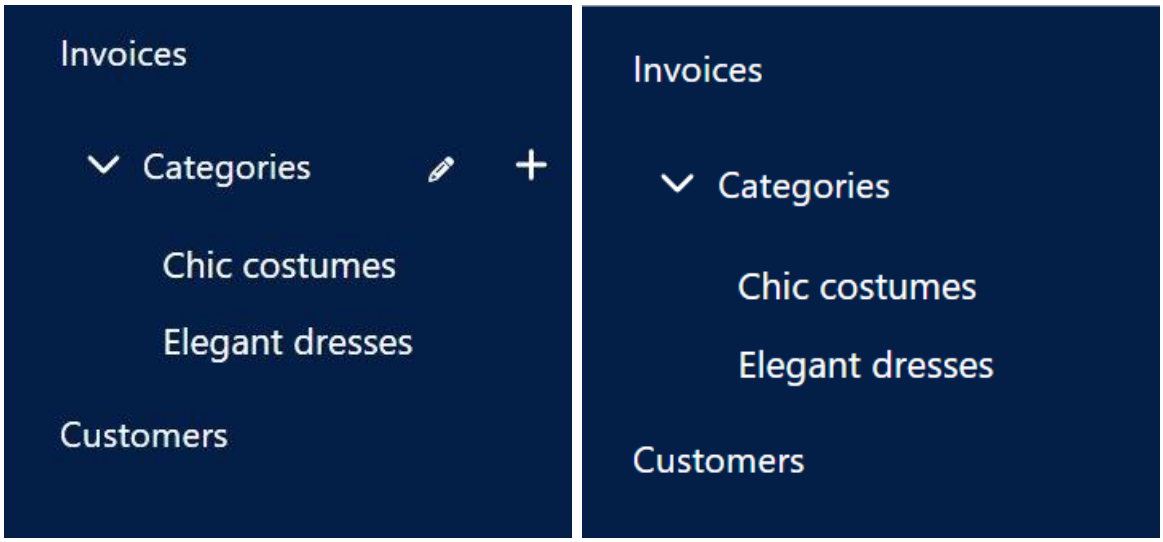


Рисунок 2.16 – Електронний лист із запрошенням працівника до реєстрації

Адміністратор на відміну від співробітника може додавати, редагувати та видаляти категорії. Цю відмінність показано на рисунку 2.18 а) та рисунку 2.18 б).



а) б)

Рисунок 2.18 – Відмінності бічної панелі для користувачів з різними дозволами:

а) бічна панель адміністратора; б) бічна панель співробітника

Користувач із дозволом «Адміністратор» має можливість створювати категорії, які можуть містити або не містити підкатегорії (рис. 2.19). Якщо він створює категорію з підкатегоріями, основна категорія діє як групування і не доступна для перегляду або додавання продуктів до неї.

Після створення категорії відображаються на лівій бічній панелі меню, доступні для перегляду, розгорнувши вкладку «Категорії» (рис. 2.20).

Щоб додати продукти до вибраної категорії, користувачу треба ввести категорію та вирішити, як додати продукти. Це можна зробити вручну, натиснувши кнопку «Додати новий продукт» у нижньому лівому куті, або імпортувавши дані з файлу, натиснувши кнопку «Імпортувати з файлу» (рис. 2.20). Адміністратор може додавати, редагувати та видаляти продукти.

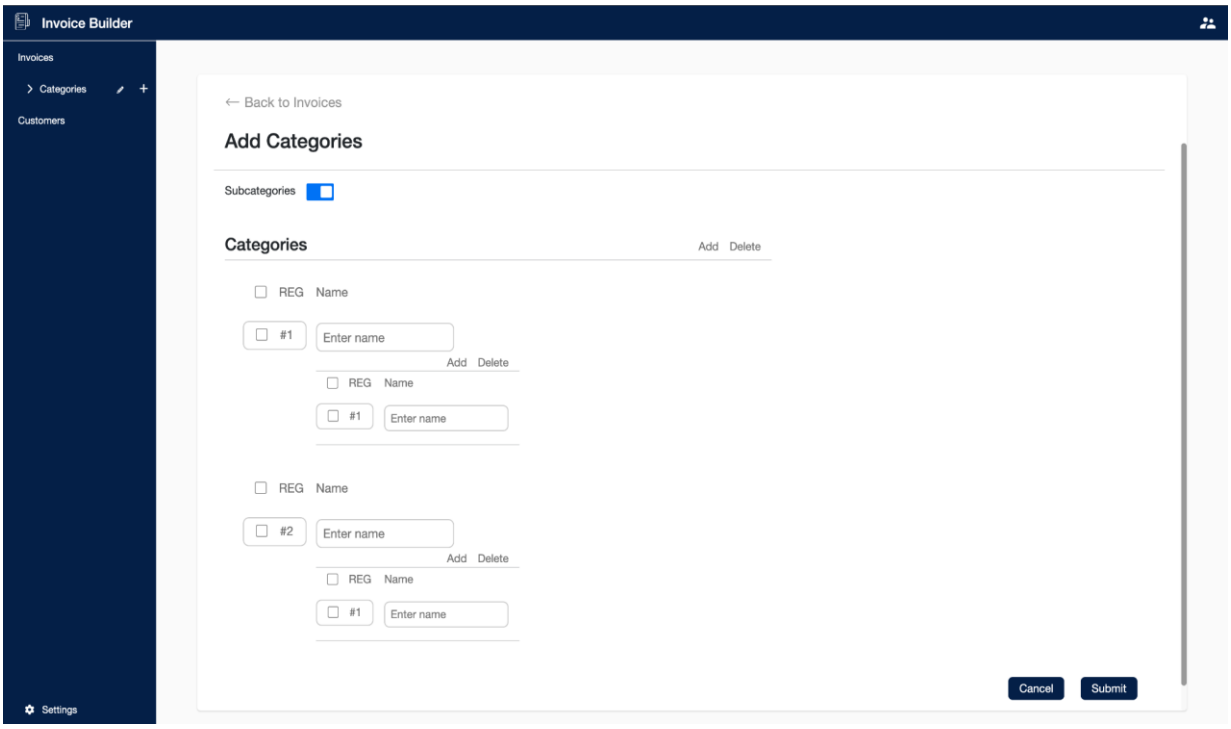


Рисунок 2.19 – Екран створення категорії з підкатегоріями

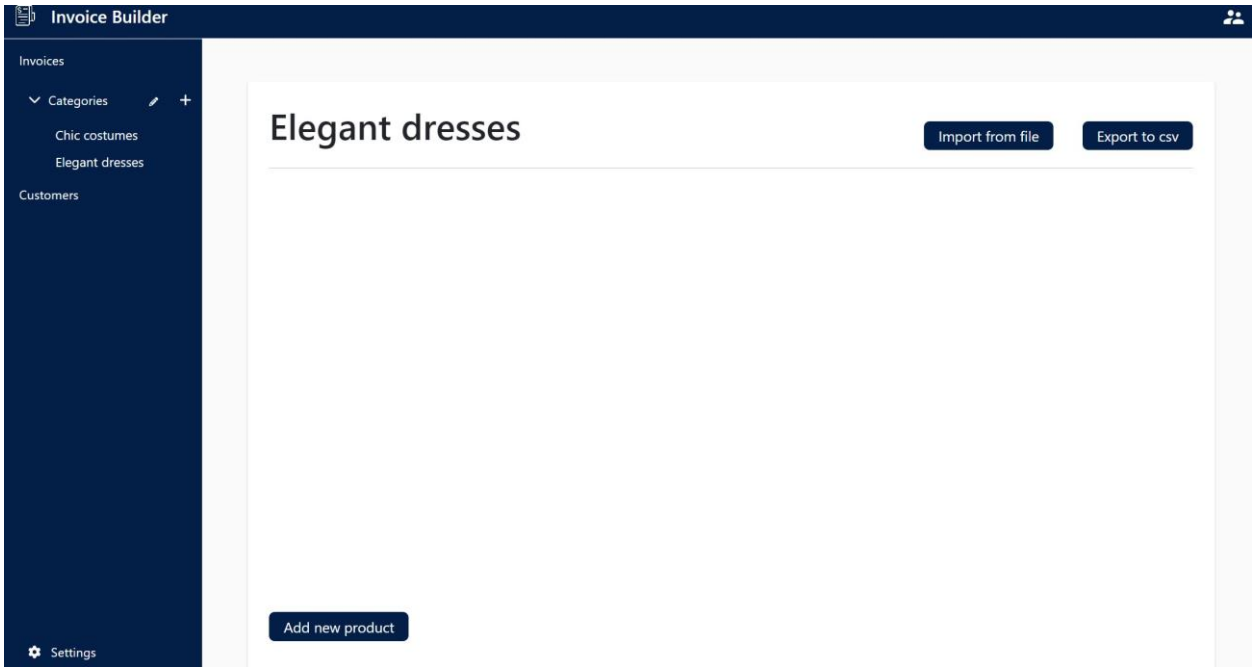


Рисунок 2.20 – Екран категорії продуктів

Після вибору опції «Імпортувати з файлу» користувач перенаправляється на сторінку «fileUpload», де він може додати файл. Якщо в завантаженому файлі більше двох аркушів, користувачеві пропонується два варіанти:

– додавання товарів тільки з одного вибраного аркуша;

– додавання продуктів з кількох аркушів, при цьому кожен аркуш розглядається як окрема категорія (рис.2.21).

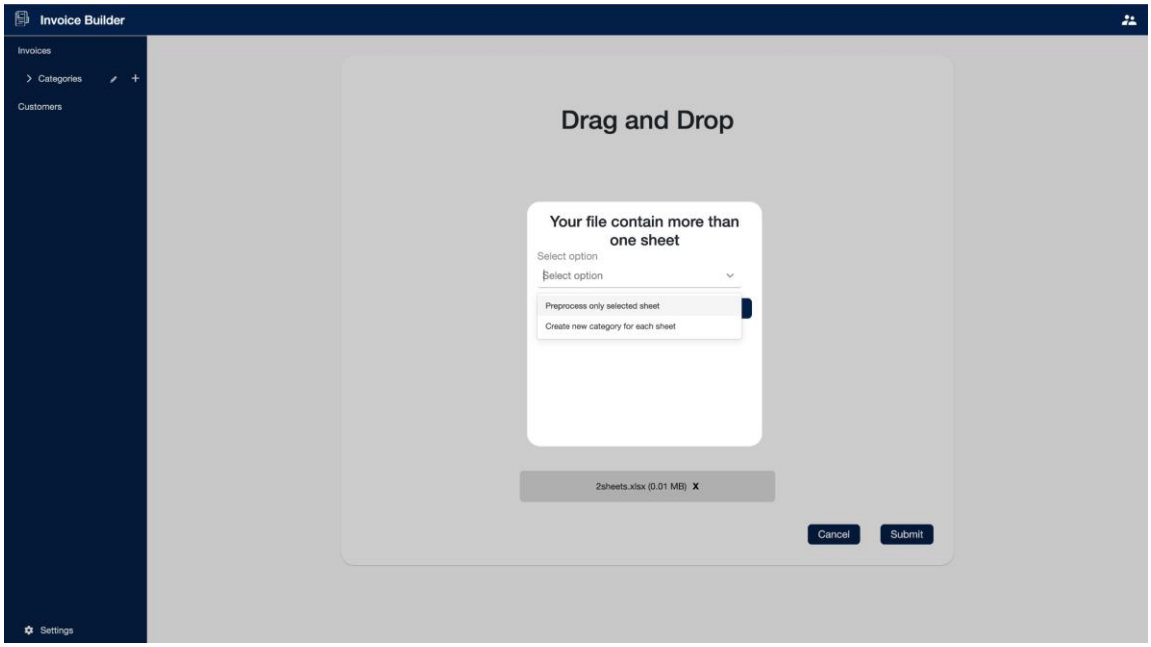


Рисунок 2.21 – Екран додавання продуктів за файлами

Після вибору вибраного параметра користувач вибирає відповідні аркуші, визначає їх у полі заголовка, а потім переходить до екрана зіставлення даних.

Екран зіставлення даних відрізняється залежно від вибраного параметра: додавання продуктів з одного аркуша або створення нової категорії для кожного аркуша. Ключовою відмінністю є наявність назв аркушів у другому варіанті та поля для введення назви для кожної новоствореної категорії. Спільним для обох екранів зіставлення є наявність полів «Назва стовпця», «Тип» і «Використовувати в рахунку». З точки зору типу, є два ключових атрибути, які повинні бути призначені кожному продукту: «назва» і «ціна». Якщо для товару не призначено поле «ціна», він не відображатиметься в списку товарів під час створення рахунку.

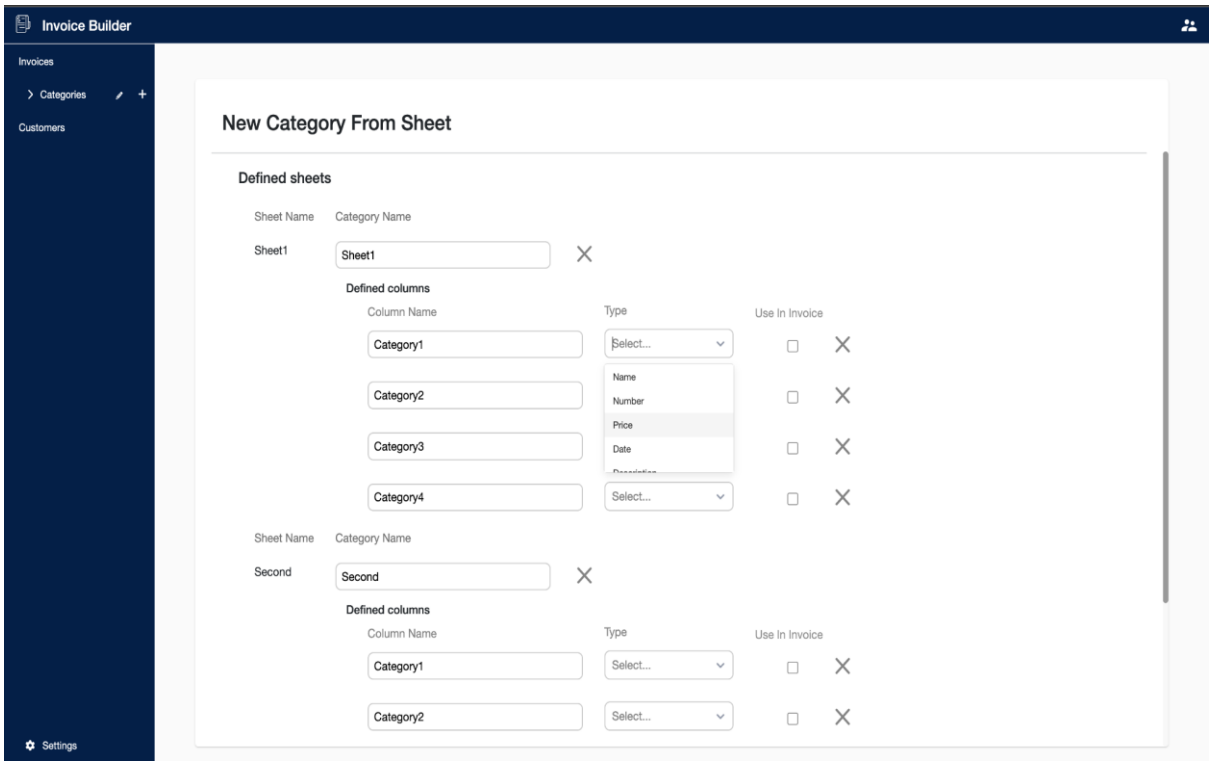


Рисунок 2.22 – Екран відображення продукту за файлами

На рисунку 2.23 зображений екран із доданими продуктами.

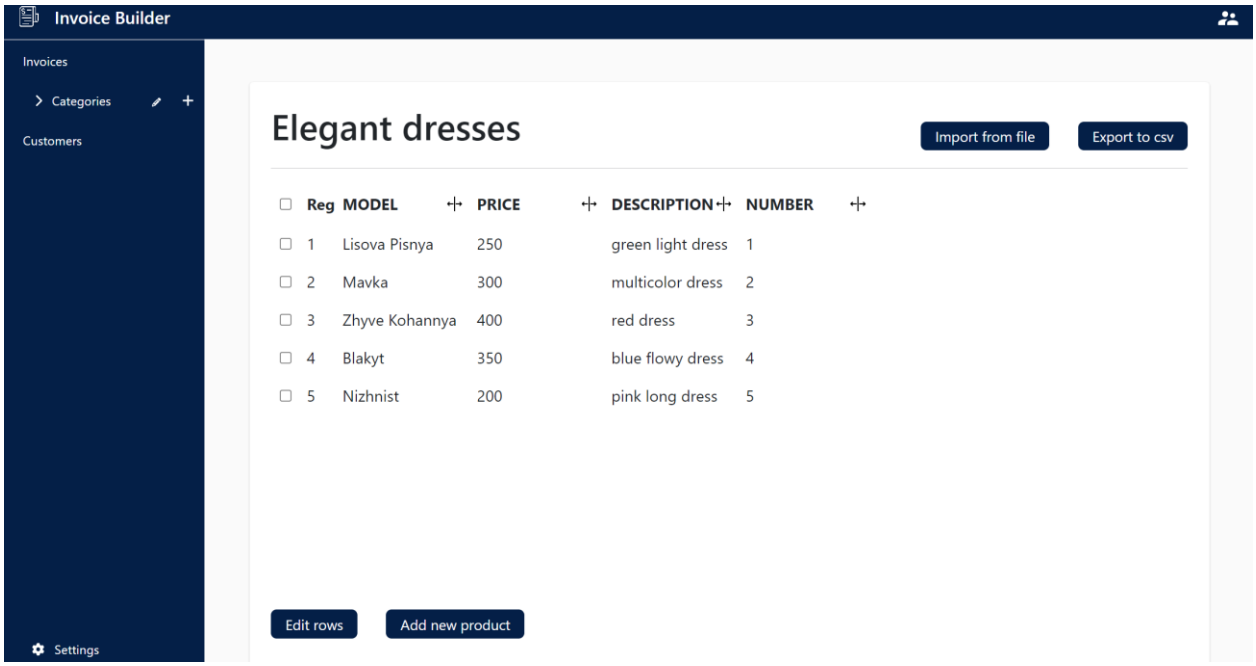


Рисунок 2.23 – Екран категорії продукту з продуктами

Кожен користувач може додати клієнта. Для цього треба перейти на вкладку «Клієнти» в бічній панелі меню та натисніть кнопку «Додати нового клієнта» (рис. 2.24).

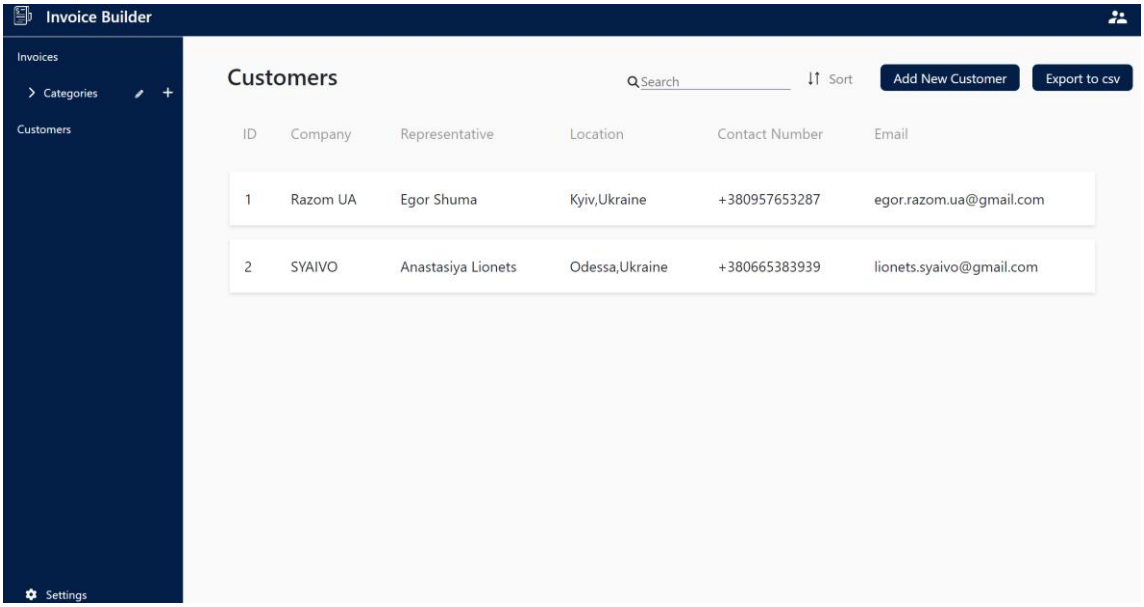


Рисунок 2.24 – Екран клієнтів

Після натискання на поле з даними клієнта система автоматично перенаправляє користувача до перегляду детальної інформації про цього клієнта, а також до списку рахунків, які йому вже виставлені (рис. 2.25).

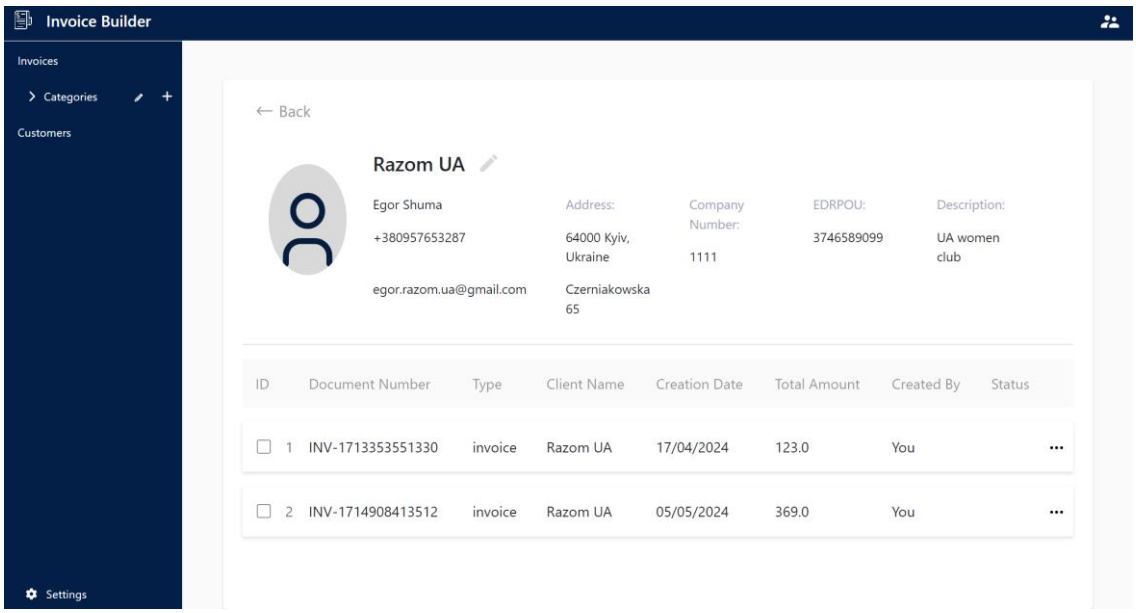


Рисунок 2.25 – Екран даних клієнта

Натиснувши рахунок-фактуру як на вкладці клієнта, так і на головній сторінці, користувач може переглянути попередній перегляд цього рахунку-фактури (рис. 2.26). Крім того, є можливість завантажити рахунок на диск або роздрукувати його.

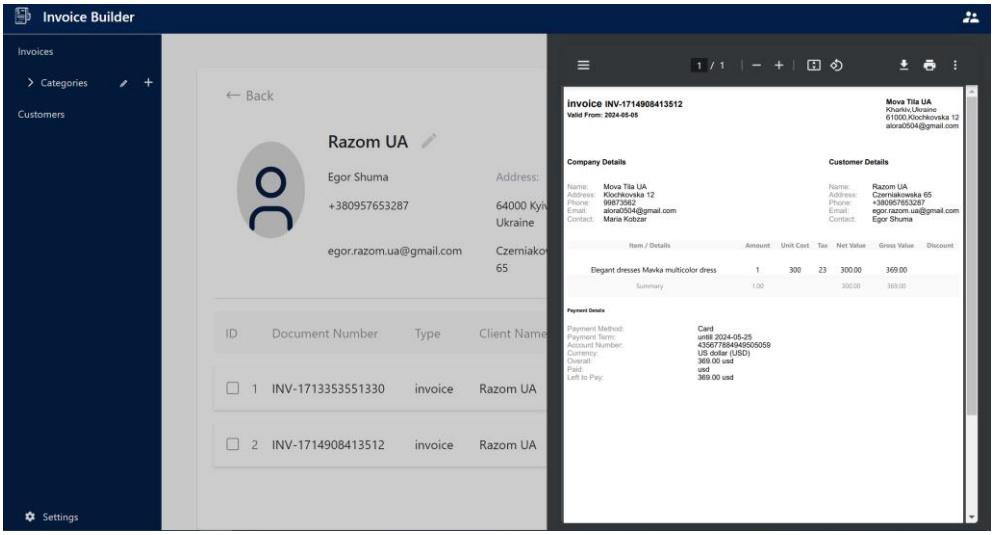


Рисунок 2.26 – Попередній перегляд рахунку-фактури

Тепер, маючи всі необхідні дані, користувач може створити рахунок. Нам потрібно повернутися на головну сторінку та натиснути кнопку «Створити новий рахунок» (рис. 2.27).

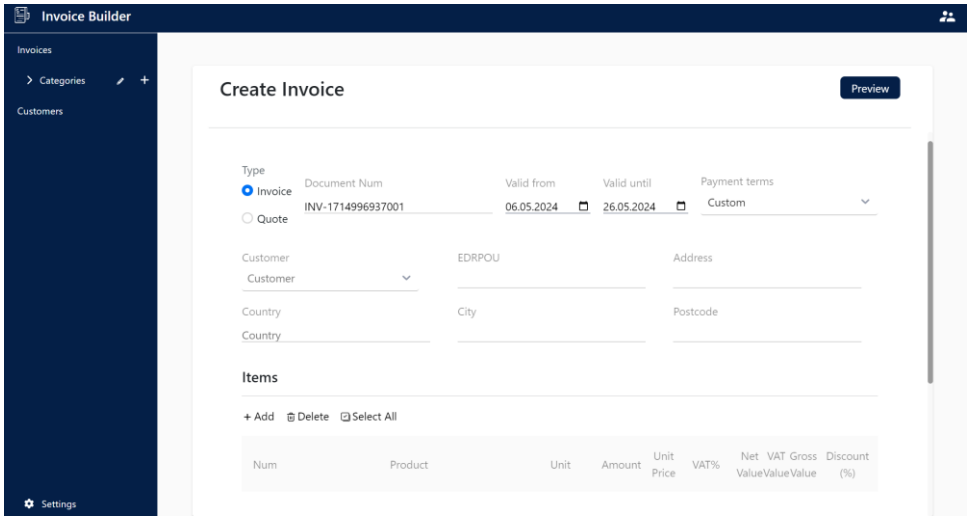


Рисунок 2.27 – Екран створення рахунку

Для оперативності заповнення рахунку було створено організацію селектора продуктів, де користувач може шукати продукти або вибирати з доступних категорій (рис. 2.28).

Це рішення є також перевагою серед запропонованих на ринку програм. Таким чином вдалося оптимізувало процес заповнення даних у документі користувачем, адже обирати зі списку зручніше ніж вводити назву вручну по пам’яті.

Рисунок 2.28 – Екран вибору продуктів

Таким чином інтерфейс програми є інтуїтивно зрозумілим та відповідає передбачуваним цілям, потребам і вимогам, зазначеним на початку процесу проектування.

3 КОМП'ЮТЕРНА МОДЕЛЬ СИСТЕМИ УПРАВЛІННЯ РАХУНКАМИ

3.1 Обґрунтування вибору технологій

В рамках кваліфікаційної роботи було реалізовано процес розробки програмного продукту, який складається з двох основних частин:

– бекенда, технічною частиною проєкту, під якою розуміється все те, що не бачить звичайний користувач, наприклад: алгоритми, обробка, робота з базою даних та інше. Основною технологією бекекнда є середовище Node.js та база даних PostgreSQL;

– фронтенда, частини, яка відповідає за графічний інтерфейс.

Основною технологією фронтенда є Next.js.

Node.js – це серверне середовище виконання JavaScript, тому воно надає всі інструменти, необхідні для запуску програм JavaScript. Node.js, заснований на движку Google JS-V8, та є середовищем виконання JavaScript на стороні сервера. Він перетворює JavaScript на машинний код і використовується в Google Chrome.

Серед переваг обраного середовища можна зазначити, що Node.js має менеджер пакетів npm, він складається з командного рядка, який взаємодіє з віддаленим реєстром. Це дозволяє користувачам використовувати та поширювати модулі JavaScript, доступні в реєстрі. Npm значно полегшує роботу з пакетами (встановлення, оновлення та видалення), а також процес розгортання проєкту на своєму комп'ютері (при роботі над проєктом у великій команді) [24-27].

Для створення системи управління рахунками було обрано цю технологію тому, що:

– у Node.js наявні різноманітні бібліотеки, які полегшують роботу. У рамках кваліфікаційної роботи знадобилася бібліотека, яка зробила роботу з

файлами Excel ефективнішою;

– Node.js і Next.js є фреймворками JavaScript, написання проєкту однією мовою збільшить продуктивність програми;

– доступні можливості Sequalize – ORM (Object-Relational Mapping) для Node.js, який підтримує кілька баз даних, включаючи PostgreSQL, MySQL, SQLite і MSSQL. Це дозволяє взаємодіяти з базою даних за допомогою JavaScript, замість написання запитів SQL.

Next.js – це фреймворк розробки вебзастосунків за допомогою бібліотеки React, який пропонує низку вдосконалень порівняно зі стандартним підходом React, наприклад рендеринг на стороні сервера (SSR) та створення статичних сторінок (SSG) або інші функції, які допомагають покращити продуктивність і оптимізувати пошукову систему.

Для управління даними було обрано базу даних PostgreSql, яка характеризується масштабованістю, надійністю та повною підтримкою SQL та JSON.

У поєднанні з серверною частиною на основі Node.js, Sequelize і PostgreSQL Next.js забезпечує цілісне сучасне рішення для створення розширених вебзастосунків, які є одночасно потужними та зручними для розробників. У контексті інженерного проєкту використання Next.js як інтерфейсного компонента дозволило створити інтерфейс користувача, який є чутливим, швидким і добре оптимізованим для пошукових систем. Ще однією великою перевагою є можливість безкоштовного розміщення інтерфейсної частини на Vercel [28-31].

Для написання роботи були використані такі інструменти:

– Figma: програма для проєкомпктування інтерфейсу та створення прототипів. Він дає змогу онлайн-дизайну UI/UX, співпраці в реальному часі та презентації прототипу.

– WebStorm: розширене середовище розробки IDE JetBrains для програмування на JavaScript. Він пропонує багато функцій, таких як

підсвічування синтаксису, рефакторинг та інтегровані інструменти налагодження [32].

– pgAdmin: інструмент керування базою даних PostgreSQL. Він дозволяє керувати, розробляти та адмініструвати бази даних у графічному інтерфейсі користувача.

– Lucidchart: вебпрограма для створення діаграм. Він дозволяє створювати різні типи діаграм, наприклад блок-схеми, діаграми ERD, інтелект-карти та інші.

– Postman: інструмент тестування та документування API, який дозволяє створювати, надсилати та аналізувати HTTP-запити в інтерактивному інтерфейсі [33].

3.2 Реалізація бекенда програми

3.2.1 Етапи впровадження серверної частини

Процес впровадження серверної частини вимагав кількох згаданих раніше попередніх кроків, таких як визначення вимог, проєктування серверної частини, налаштування середовища розробки, впровадження серверної частини, тестування, розгортання серверної частини, моніторинг і підтримка роботи серверної частини.

Нижче покроково описаний процес створення бекенда:

Крок 1. Налаштування проєкту: створити новий проєкт у Node.js, ініціалізувавши новий каталог за допомогою npm init і налаштувавши основні залежності.

Крок 2. Конфігурація бази даних: налаштувати базу даних PostgreSQL і бібліотеку Sequelize для підключення до бази даних.

Для цього було створено екземпляр Sequelize, який передав URL-адресу бази даних, визначену в змінних середовища як process.env.DATABASE_URL

і додаткові параметри конфігурації. Фрагмент конфігурації показаний на рисунку 3.1.

```
const Sequelize :  = require('sequelize');

const sequelize = new Sequelize(process.env.DATABASE_URL, {
    dialect: 'postgres',
    protocol: 'postgres',
    dialectOptions: {
        ssl: {
            require: true,
            rejectUnauthorized: false
        }
    }
});
module.exports = sequelize;
```

Рисунок 3.1 – Фрагмент конфінурації з файлу database.js

Крок 3. Визначення моделі даних: створити моделі за допомогою Sequelize, які представляють сутності або об’єкти на сервері, такі як користувачі, замовлення або продукти.

На основі діаграми зв’язку сутностей, що зображена на рисунку 2.2, виконано визначення моделі даних. Для цього створено класи, які представляють сутності або об’єкти на сервері.

На рисунку 3.2 показано, що клас Category було визначено за допомогою методу define , наданого Sequelize. Він описує сутність категорії, яка має такі атрибути:

- id: автоматично збільшений унікальний ідентифікатор для кожної категорії;
- name: назва категорії, обов’язкове поле;
- description: додатковий опис категорії;
- parentId: зовнішній ключ, що вказує на батьківську категорію. Якщо немає батьківської категорії, вона має значення null . Видалення або оновлення батьківської категорії каскадно передаються до нащадків.

```
const Sequelize :   = require('sequelize');

const sequelize :{}  = require('../util/database');

const Category = sequelize.define('Category', {
    id: {
        type: Sequelize.INTEGER,
        autoIncrement: true,
        allowNull: false,
        primaryKey: true
    },
    name: {
        type: Sequelize.STRING,
        allowNull: false
    },
    description: Sequelize.STRING,
    parentId: {
        type: Sequelize.INTEGER,
        allowNull: true,
        references: {
            model: 'Categories',
            key: 'id'
        },
        onDelete: 'CASCADE',
        onUpdate: 'CASCADE'
    }
});

module.exports = Category;
```

Рисунок 3.2 – Фрагмент створення сутності категорії товарів з файлу category.js

За допомогою Sequelize створення та керування цією моделлю стає простим і не вимагає безпосереднього написання запитів SQL. Щоб створити зв’язки між сутностями, Sequelize надає такі методи, як hasMany, belongsTo тощо, які дозволяють визначати зв’язки між різними моделями. На рисунку 3.3 наведено приклад створення зв’язку.

Крок 4. Створення контролерів: визначити контролери, які оброблятимуть HTTP-запити та пересилатимуть їх на рівень обслуговування.

В архітектурі програми на основі Node.js, що використовує Express.js, контролери (представлені як «routes») відповідають за обробку HTTP-запитів і делегування відповідальності відповідним функціям (відомим як

«middleware») для реалізації бізнес-логіки. Щоб визначити контролери в програмі Express.js, використовуйте методи, надані об’єктом маршрутизатора, наприклад get , post , put.

```
// CATEGORY TO CATEGORY
db.category.hasMany(db.category, options: { as: 'subCategories', foreignKey: 'parentId' });
db.category.belongsTo(db.category, options: { as: 'parent', foreignKey: 'parentId' });

// CATEGORY & COMPANY
db.company.hasMany(db.category, options: {onDelete: 'CASCADE', hooks: true});
db.category.belongsTo(db.company);

//CATEGORY & PRODUCT
db.category.hasMany(db.product, options: {onDelete: 'CASCADE', hooks: true});
db.product.belongsTo(db.category);
```

Рисунок 3.3 – Фрагмент коду створення зв’язків між моделями з файлів models та index.js

На рисунку 3.4 показано, що URL-шляхи визначено за допомогою router.get для запитів GET і router.post для запитів POST. Кожен шлях пов’язаний із певною функцією контролера (наприклад, categoryController.addCategories), яка відповідає за виконання даної операції.

```
const express : e | () => core.Express  = require('express');
const categoryController : {...}  = require("../controllers/categoryController");
const {authorize} = require("../middleware/authJwt");

const router : Router  = express.Router();

router.get( path: "/getCategoriesWithSubcategories/:CompanyId", authorize, categoryController.getCategoriesWithSubcategories)

router.post( path: "/addCategories/:CompanyId", authorize, categoryController.addCategories);

router.delete( path: "/deleteCategory/:CategoryId", authorize, categoryController.deleteCategory);

router.post( path: "/updateCategory/:CategoryId", authorize, categoryController.updateCategory);


module.exports = router;
```

Рисунок 3.4 – Контролер категорії з файлу categoryRoutes.js

Крок 5. Створення сервісів: визначення сервісів, які оброблятимуть бізнес-логіку та взаємодію з базами даних за допомогою Sequelize. Сервіси використовуються для реалізації бізнес-логіки програми та зазвичай використовуються контролерами (рис. 3.5).

```
exports.deleteCategory = [validateRequest( requiredParams: ['CategoryId'], requiredBody: []), async (req, res, next) : Promise<void>  => {
    await IdVerifications.categoryExists( {CategoryId}: {CategoryId: req.params.CategoryId});
    try {
        await Category.destroy( options: {
            where: {
                id: req.params.CategoryId,
            }
        })

        res.send({
            message: "Category was deleted successfully!"
        });
    } catch (err) {
        next(err);
    }
}]

exports.updateCategory = [validateRequest( requiredParams: ['CategoryId'], requiredBody: ['name']), async (req, res, next) : Promise<void>  => {
    await IdVerifications.categoryExists( {CategoryId}: {CategoryId: req.params.CategoryId});
    try {
        let category : {description: any, name: any}  = {
            name: req.body.name, description: req.body.description,
        }
        await Category.update(category, {
            where: {
                id: req.params.CategoryId
            }
        })
        res.send({
            message: "Category was updated successfully!"
        });
    } catch (err) {
        next(err);
    }

}]
```

Рисунок 3.5 – Сервіс для категорій товарів (categoryController.js)

У системі управління рахунками сервіс категорій складається з таких функцій, як updateCategory , deleteCategory , які відповідають за певні операції з категоріями: оновлення категорій та видалення категорій.

Кожна функція сервісу відповідає за певну бізнес-логіку, наприклад перевірку вхідних даних, взаємодію з базою даних за допомогою ORM (у цьому випадку Sequelize) або підтримку різних бізнес-сценаріїв.

Крок 6. Тестування бекенда: перевірити бекенд, вручну надсилаючи HTTP-запити до API або використовуючи такі інструменти тестування, як Postman (рис. 3.6) [33,34].

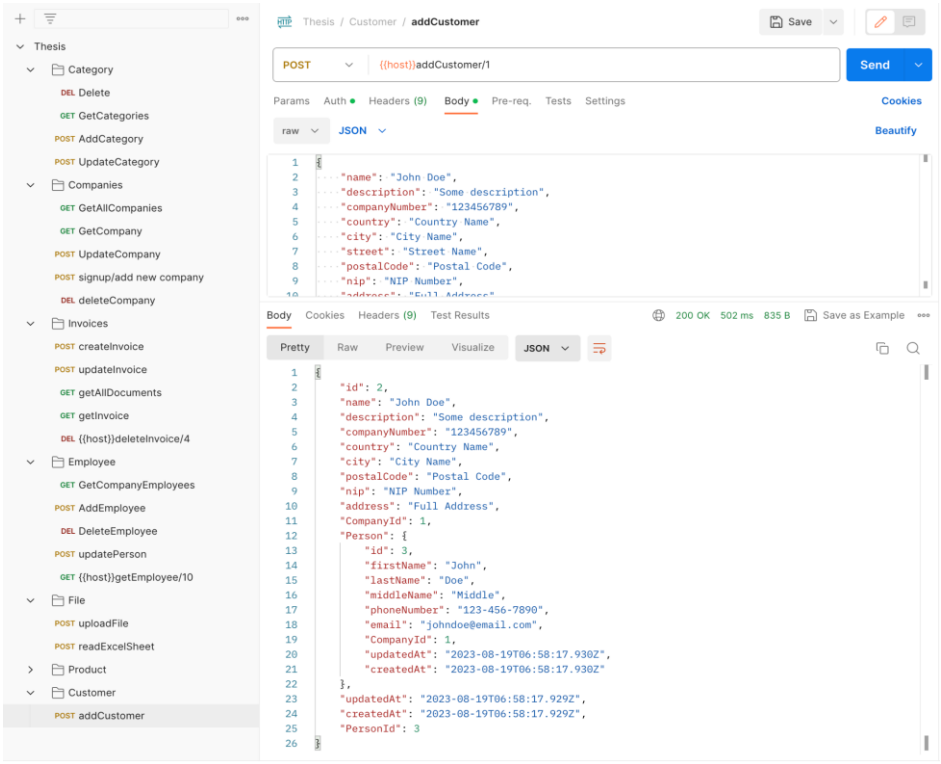


Рисунок 3.6 – Бекенд-тестування за допомогою Postman

3.2.2 Проблеми під час впровадження бекенду та їх вирішення

Однією з ключових технічних проблем, що виникли під час впровадження, було ефективне зберігання продуктів із різними характеристиками в базі даних. Продукти можуть мати різні атрибути, які важко моделювати в традиційних реляційних базах даних, наприклад, в Oracle, Microsoft SQL Server, MySQL.

Спочатку розглядалося використання динамічних баз даних для моделювання цих змінних атрибутів продукту. Однак вибір пав на PostgreSQL, тому що вона пропонує функціональність, яка відповідає вимогам проєкту - можливість використовувати атрибути JSONB. Завдяки цьому функціоналу

вдалося створити модель продукту, в якій більшість атрибутів є постійними, а змінна структура атрибутів продукту зберігається в одному полі JSONB [35].

Лістинг 3.1 ілюструє ключовий фрагмент моделі продукту, який показує використання цієї функціональності:

other: Sequelize . *JSONB* ,

Інше поле типу JSONB дозволяє зберігати будь-яку структуру даних у вигляді об’єкта JSON. Це рішення дозволило гнучко моделювати різні атрибути продукту, зберігаючи продуктивність і структурну цілісність бази даних.

Ще одним викликом було забезпечити ефективне зберігання рахунків. Користувачам необхідно завантажити або роздрукувати створені рахунки, що вимагає швидкого доступу до них. Ця проблема була вирішена за допомогою об’єктно-орієнтованої служби зберігання Amazon S3, що забезпечило безпеку та безперебійну доступність до рахунків [36].

Файли рахунків надсилаються в Amazon S3 таким чином, як показано на рисунку 3.7, а реалізація отримання файлу на рисунку 3.8.

```
const amazonS3 : S3  = new AWS.S3( options: {
    accessKeyId: process.env.ACCESS_KEY_ID,
    secretAccessKey: process.env.SECRET_ACCESS_KEY,
});

exports.createFile = async (file, id) : Promise<...>  => {
    try {
        const { path, filename } = file;
        const uniqueFilename : string  = `${uuidv4()}-${filename}`; // Generate a unique filename

        const params : {...}  = {
            Bucket: process.env.S3_BUCKET,
            Key: uniqueFilename,
            Body: fs.readFileSync(path),
        };

        const data : SendData  = await amazonS3.upload(params).promise();
        fs.unlinkSync(path);

        return data;
    } catch (error) {
        fs.unlinkSync(file.path);
        throw new Error(error);
    }
}
```

Рисунок 3.7 – Фрагмент коду завантаження файлу в S3, файл amazonS3.js

```
31  exports.getFileFromS3 = async (url) : Promise<...>  => {
32      const Key = url.split("/").pop();
33      const params : {Bucket: any, Key: unknown}  = {
34          Bucket: process.env.S3_BUCKET,
35          Key : Key,
36      };
37
38      try {
39          const data : GetObjectOutput & {...}  = await amazonS3.getObject(params).promise();
40          return data.Body;
41      } catch (error) {
42          console.error(`Failed to retrieve file from S3: ${error}`);
43          return null;
44      }
45  };
```

Рисунок 3.8 – Фрагмент коду отримання файлів з amazon s3, файл amazonS3.js

Завдяки механізму описаному нижче вдалося значно підвищити рівень безпеки та надійності процесу реєстрації користувачів. Ключовим елементом цього процесу став механізм підтвердження електронної адреси користувача. Для вирішення цього завдання була використана бібліотека Nodemailer. Ця бібліотека пропонує інструменти для надсилання електронних листів за допомогою Node.js. У цьому випадку Nodemailer використовувалась для створення та надсилання електронних листів із посиланням для підтвердження реєстрації користувача.

Для забезпечення безпеки та оптимізації ресурсів важливо регулярно видаляти старі або прострочені токени. Тому було створено сценарій для видалення прострочених маркерів запрошення (старше 1 години), що показаний на рисунку 3.9.

```
function deleteExpiredTokens() : void  {
    let EXPIRATION_TIME : number  = 86400000; // 24 hours in milliseconds
    const expirationDate : Date  = new Date( value: Date.now() - EXPIRATION_TIME);
    Invitation.destroy( options: {
        where: {
            createdAt: {
                [Op.lt]: expirationDate
            }
        }
    })
        .then(() : void  => {
            console.log('Expired tokens deleted');
        })
        .catch(err => {
            console.error('Error deleting tokens', err);
        });
}

// Then schedule the job to run every hour
cron.schedule( expression: '0 * * * *', deleteExpiredTokens);
```

Рисунок 3.9 – Видалення прострочених токенів із бази даних, файл app.js

3.3 Програмна реалізація фронтенду

Реалізація інтерфейсу передбачає розробку макета та зовнішнього вигляду програми, впровадження коду HTML, CSS і JavaScript, щоб перетворити дизайн на робочий вебсайт, а також інтеграцію інтерфейсу з серверною частиною програми.

Next.js — популярний фреймворк для створення застосунків на основі React, який дозволяє створювати універсальні застосунки, тобто працювати на стороні сервера та клієнта. Щоб реалізувати інтерфейс програми за допомогою Next.js, потрібно встановити інструменти Next.js, створити проєкт і реалізувати код HTML, CSS (у рамках кваліфікаційної роботи за допомогою module.css) і JavaScript.

Нижче покроково розписано процес впровадження інтерфейсу програми:

Крок 1. Встановлення Next.js і налаштування проєкту. Це можна зробити через термінал, виконавши команду: npm install next react react-dom. Наступним кроком є створення проєкту та встановлення необхідних

залежностей. Перейдіть до відповідного каталогу та виконайте команди інтерфейсу npx create-next-app і npm install.

Створення файлу конфігурації є ще одним важливим кроком. За замовчуванням для Next.js не потрібен окремий файл конфігурації, але якщо треба налаштувати певну поведінку фреймворку, то можна створити файл next.config.js у корені проєкту.

```
1  /** @type {import('next').NextConfig} */
2  const nextConfig : NextConfig  = {
3    reactStrictMode: true,
4    env: {
5      API_URL: process.env.API_URL,
6    }
7  }
8
9  module.exports = nextConfig
```

Рисунок 3.10 – Файл конфігурації (next.config.js)

Після виконання вищевказаних кроків рекомендується виконати команду npm run dev, щоб запустити проєкт і перевірити його роботу. Таким чином можна перевірити, чи всі компоненти програми працюють належним чином. На рисунку 3.11 наведено результат налаштування проєкту в Next.js.

Рисунок 3.11 – Структура проєкту Next.js після налаштування

Крок 2. Створення відповідних компонентів React, які представлятимуть інтерфейс користувача [37].

Компоненти сторінки зберігаються в папці pages. Наприклад, компонент pages/customers.js або pages/customers/index.js буде доступним у шляху /customers . Спільні компоненти: верхні, нижні колонтитули та бічні панелі навігації, можна зберігати в папці компонентів і імпортувати у відповідні компоненти сторінки, щоб уникнути повторення коду. За допомогою module.css стиль у Next.js стає простим і модульним. Можна створити окремий файл CSS для кожного компонента та імпортувати його в компонент, дозволяючи ізолювати стиль.

Крок 3. Створення логіки та представлення відповідної взаємодії та функцій, доступних для користувача.

У Next.js ключовим елементом створення логіки є «хуки», які є функціями, які дозволяють використовувати функціональні можливості React у функціональних компонентах. Хоча Next.js не використовує традиційну концепцію «сервісів», відому з Angular, можна створювати власні хуки, які виконують подібну функцію — вони дозволяють ізолювати та повторно використовувати бізнес-логіку в різних компонентах.

Фрагмент коду на рисунку 3.12 демонструє використання React Hooks, які дозволяють ефективно керувати станом і життєвим циклом компонента.

За допомогою хука useState визначено кілька змінних стану. Ці змінні зберігають інформацію, серед іншого: про адміністраторів, користувачів, посилання для реєстрації та статус різних елементів інтерфейсу. Такі функції, як setAdmins і setRegisterLink, дозволяють оновлювати ці значення у відповідь на різні події в програмі.

На наступному етапі впровадження використовувався хук useEffect, який активується після впровадження компонента. У його тілі визначено асинхронну функцію fetchData , основним завданням якої є зв’язок із внутрішнім інтерфейсом API, який, у свою чергу, спілкується із зовнішньою

службою. Після отримання відповіді дані обробляються та оновлюються за допомогою попередньо визначених сеттерів, що призводить до оновлення інтерфейсу користувача.

```
const Accounts = () => {

    const [admins : [] , setAdmins] = useState( initialState: [])
    const [users : [] , setUsers] = useState( initialState: [])
    const [registerLink : string , setRegisterLink] = useState( initialState: '')
    const [showConfirmationDialogBeforeUserDelete : boolean , setShowConfirmationDialogBeforeUserDelete] = useState( initialState: false)
    const [deleteUserId : string , setDeleteUserId] = useState( initialState: '')
    const [email : string , setEmail] = useState( initialState: '')
    const [emailValid : boolean , setEmailValid] = React.useState( initialState: true);
    const [emailMessage : string , setEmailMessage] = React.useState( initialState: '');
    const [copySuccess : string , setCopySuccess] = useState( initialState: '');


    useEffect( effect: () : void  => {
        1 usage
        const fetchData = async () : Promise<void>  => {
            const data : ... = await getEmployees()

            const token = await getRegisterToken()

            const admins : any[] = data.employees.filter(employee : T  =>
                employee.Permissions.some(permission : {name: any}  => permission.name === 'admin')
            )
            const users : any[] = data.employees.filter(employee : T  =>
                employee.Permissions.every(permission : {name: any}  => permission.name !== 'admin')
            )

            setAdmins(admins)
            setUsers(users)
            setRegisterLink( value: `https://invoice-builder-blue.vercel.app/userSignUp/${token.data.token}`)
        }

        fetchData()
    }, deps: [])

```

Рисунок 3.12 – Використання хуків React, файл accounts.js

Крок 4. Бекенд-інтеграція. Створення функцій API та їх зв’язок із відповідними компонентами для обміну даними між інтерфейсом і бекендом.

Серверна інтеграція відіграє життєво важливу роль в архітектурі застосунків клієнт-сервер. У цьому контексті була використана бібліотека axios, яка використовується для виконання асинхронних HTTP-запитів на основі об’єктів типу Promise. Завдяки своєму дизайну axios пропонує зрозумілий інтерфейс програмування для взаємодії з різними HTTP-сервісами.

На рисунку 3.13 показано фрагмент коду, де:

– baseConfig : визначена конфігурація, яка встановлює базову URL-адресу для кінцевих точок API та стандартний заголовок Content-Type у

application/json. Це означає, що зв’язок із сервером здійснюватиметься у форматі JSON.

– api : це стандартний екземпляр axios , призначений для виконання неавтентифікованих запитів.

– authorizedApi : екземпляр axios , налаштований для обробки запитів,які потребують авторизації. Використовуючи інтерфейс перехоплювача, цей екземпляр може динамічно приєднувати заголовок авторизації до кожного запиту.

```
import axios from 'axios';
import {getCookie} from "cookies-next";

const baseConfig : {baseURL: string, headers: {...}}  = {
    baseURL: process.env.API_URL,
    headers: {
        'Content-Type': 'application/json',
    },
};

// API instance for non-authorized requests
no usages
const api = axios.create(baseConfig);

// API instance for authorized requests
5+ usages
const authorizedApi = axios.create(baseConfig);

// Using an interceptor to attach the Authorization header dynamically to the authorizedApi instance
authorizedApi.interceptors.request.use( onFulfilled: config : InternalAxiosRequestConfig  => {
    const token : CookieValueTypes  = getCookie( key: "accToken");
    if (token) {
        config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
});

export { api, authorizedApi };
```

Рисунок 3.13 – Конфігурація Axios, файл api.js

На рисунку 3.14 проілюстровано функції API, які показують, побудову комунікаційного інтерфейсу:

– companySignup: реєструє компанію в системі;

– login: Автентифікує користувача в системі;

– getRegisterToken: отримує спеціальний маркер, який використовується під час процесу реєстрації;

– employeeSignUp: реєструє працівника з наданим маркером;

– sendRegisterLinkViaEmail: надсилає реєстраційне посилання на вказану адресу електронної пошти через захищену кінцеву точку.

```
export const login = async (data) : Promise<...>  => {
    try {
        return await api.post( url: `signIn`, data);
    } catch (error) {
        console.error("Error in login process:", error.response || error);
    }
};

2 usages
export const getRegisterToken = async () : Promise<...>  => {
    try {
        return await api.get( url: `createInvite/${getCookie( key: "companyId")}`);
    } catch (error) {
        console.error("Error in login process:", error.response || error);
    }
};

2 usages
export const employeeSignUp = async (data, token) : Promise<...>  => {
    try {
        return await api.post( url: `employeeSignup/${token}`, data);
    } catch (error) {
        console.error("Error in login process:", error.response || error);
    }
};

2 usages
export const sendRegisterLinkViaEmail = async (email) : Promise<...>  => {
    try {
        return await authorizedApi.post( url: `sendRegisterLinkViaEmail/${getCookie( key: "companyId")}`, email);
    } catch (error) {
        console.error("Error in login process:", error.response || error);
    }
};
```

Рисунок 3.14 – Методи API, файл authorizationApi.js

Після завершення вищезазначених дій програму можна запускати локально командою npm run dev або розгортати на вебсервері для виконання тестів на прийняття, наприклад, у браузері.

Програма Next.js створена та підтримується локально запуском команди npm run dev. Ця команда створює програму Next.js і запускає сервер розробки для розміщення програми. Сервер розробки автоматично перезавантажує програму, коли в код вносяться зміни, що полегшує розробку та тестування програми. За замовчуванням сервер розробки буде доступний за адресою http://localhost:3000/, а програму можна переглянути у веббраузері, перейшовши за цією URL-адресою.

ВИСНОВКИ

У рамках кваліфікаційної роботи був розроблений і реалізована система управління рахунками за допомогою таких мов програмування: JavaScript, CSS, HTML та сучасних технологій Node.js і Next.js.

Реалізовано два види доступів «Адміністратор» та «Співробітник», що забезпечить можливостями дбати про захист даних компанії, оптимізувати робочий процес, підвищити рівень відповідальності співробітників та відстежувати їх продуктивність.

Створена система дозволяє користувачам-керівникам підприємств підключати працівників до системи за допомогою автоматизованого рішення запрошення співробітника до реєстрації та підтвердження реєстрації електронною поштою, що дозволяє корпоративно налаштовувати застосунок онлайн. Автоматичне заповнення системою даних компанії при формуванні документу зменщує ризик помилок користувача. А додавання списку товарів за допомогою Excel оптимізує роботу користувача, мінімізуючи часові витрати на ручне введення даних.

При створенні системи були враховані всі плюси та мінуси систем-аналогів, вибраних для порівняння. Проведено всі необхідні аналітичні процеси для проєктування та програмування системи та бази даних.

Вебзастосунок відповідає сучасним стандартам якості вигляду клієнтського інтерфейсу та вимогам до вигляду рахунку як документу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. ECommerce Growth from 2010 to 2020. URL: https://redstagfulfillment.com/2010s-ecommerce-growth-decade/ (дата звернення 10.04.2024).

2. Стан ринку e-commerce в Україні: Микита Артемчук про тенденції, виклики та точки для розвитку. URL: https://web-promo.ua/ua/blog/stan-rinku-e-commerce-v-ukrayini-mikita-artemchuk-pro-tendenciyi-vikliki-ta-tochki-dlya-rozvitku/ (дата звернення 10.04.2024).

3. Role of Entrepreneurship in Economic Development 2024. URL: https://www.nexford.edu/insights/role-of-entrepreneurship-in-economic-growth (дата звернення 10.04.2024).

4. A Brief History of Invoices. URL: https://sufio.com/blog/history-of-invoices/ (дата звернення 10.04.2024).

5. What Are the e-Invoice Requirements in Ukraine (Full Guide). URL: https://www.storecove.com/blog/en/what-are-the-e-invoice-requirements-in-ukraine (дата звернення 15.04.2024).

6, Гороховатський, В. О., & Творошенко, І. С. (2021). Методи інтелектуального аналізу та оброблення даних: навч. посібник.

7. Tvoroshenko, I., & Kharchenko, A. (2021). Some aspects of modern development for sign language recognition systems.

8. Shafronenko, A., Bodyanskiy, Y. V., Klymova, I., & Holovin, O. (2020, May). Online credibilistic fuzzy clustering of data using membership functions of special type. In CMIS (pp. 744-753).

9. Shafronenko, A., Bodyanskiy, Y., Pliss, I., & Irina, K. (2021, September). Online Credibilistic Fuzzy Clustering Method Based on Cauchy Density Distribution Function. In 2021 11th International Conference on Advanced Computer Information Technologies (ACIT) (pp. 704-707). IEEE.

10. Shafronenko, A., Bodyanskiy, Y., & Rudenko, D. (2020). Neuro-fuzzy clustering of Distorted Data Using Cat Swarm Optimization. LAP LAMBERT Academic Publishing.

11. Сучасний підручник з JavaScript. URL: https://uk.javascript.info/ (дата звернення 26.04.2024).

12. Haverbeke, M. (2018). Eloquent javascript: A modern introduction to programming. No Starch Press.

13. What is Microsoft SQL Server? A definition from Whatls.com. URL: https://www.techtarget.com/searchdatamanagement/definition/SQL-Server (дата звернення 28.04.2024).

14. Microsoft SQL Server. (2018). Admin Kit. Birn, Jeffrey. M.: LORI, 211.

15. Творошенко, І.С. (2018). Дослідження особливостей побудови нечітких відношень під час відображення динамічних взаємодіючих нечітких процесів складних систем.

16. Архітектура вебзастосунків - як вибрати?: стаття з блогу IT-школи Hillel. URL: https://blog.ithillel.ua/articles/web-application-architecture (дата звернення 29.04.2024).

17. What are Functional Requirements: Examples, Definition, Complete Guide. URL: https://visuresolutions.com/blog/functional-requirements (дата звернення 30.04.2024).

18. Що таке база даних? URL: http://apeps.kpi.ua/shco-take-basa-danykh (дата звернення 25.04.2024).

19. Lucidchart. URL: https://www.lucidchart.com (дата звернення 19.04.2024).

20. Figma for prototyping. URL: https://www.figma.com/prototyping/ (дата звернення 22.04.2024).

21. Дизайн-система державних сайтів України. URL: https://design.gov.ua/ua (дата звернення 20.04.2024).

22. Online J Commun Media Technol(2016 Jul) Literature Review: Website Design and User Engagement, 6(3), pp. 1–14.

23. Ndukwe, W. (2019). Website Builders: A Tool In Web Design From A Graphic Design Perspective.

24. 10 Most Popular Software Architectural Patterns. URL: https://nixunited.com/blog/10-common-software- architectural-patterns-part-1/ (дата звернення 01.05.2024).

25. Node.js. URL: https://nodejs.org/en/about (дата звернення 01.05.2024).

26. Herron, D. (2020). Node. js Web Development: Server-side web development made easy with Node 14 using practical examples. Packt Publishing Ltd.

27. Архітектура вебзастосунків – Telegraph. URL: https://telegra.ph/Arh%D1%96tektura-veb-dodatk%D1%96v-04-17 (дата звернення 03.05.2024).

28. Pereira, C. R., & Pereira, C. R. (2016). Working with SQL Databases. Building APIs with Node. js, 27-36.

29. Sud, K., & Sud, K. (2020). Database Connectivity. Practical hapi: Build Your Own hapi Apps and Learn from Industry Case Studies, 63-84.

30. PostgreSQL. URL: https://www.postgresql.org/ (дата звернення 25.04.2024).

31. Порівняльний аналіз популярних JavaScript - фреймворків та бібліотек для front-end розробки. URL: https://openarchive.nure.ua/entities/publication/a0cfae50-6170-4c9b-9e97-1e17ff020682 (дата звернення 25.04.2024).

32. WebStorm. URL: https://www.jetbrains.com/webstorm/ (дата звернення 23.04.2024).

33. Postman URL: https://www.postman.com/ (дата звернення 23.04.2024).

34. Tvoroshenko, I.S., & Maksimenko, H. (2021). To the question of analysis of existing mechanisms of web application testing.

35. SQLite vs MySQL vs PostgreSQL: A Comparison Of Relational Database Management System. URL: https://www.digitalocean.com/community/tutorials/sqlite-vs-mysql-vs-postgresqla-

comparison-of-relational-database-management-systems (дата звернення 08.05.2024).

36. Amazon S3. URL: https://aws.amazon.com (дата звернення 02.05.2024).

37. Banks, A., & Porcello, E. (2020). Learning React: modern patterns for developing React apps. O'Reilly Media.

Elrom, E., & Elrom, E. (2021). React Router and Material-UI. React and Libraries: Your Complete Guide to the React Ecosystem, 79-113.