МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ФАХОВИЙ КОЛЕДЖ ІНЖЕНЕРІЇ ТА УПРАВЛІННЯ
НАЦІОНАЛЬНОГО АВІАЦІЙНОГО УНІВЕРСИТЕТУ»

ДОПУСТИТИ ДО ЗАХИСТУ

Заступник директора з НР

_____ О.В. Родіонова

«____»_____ 2021 р.

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ

«БАКАЛАВР»

Тема: Розробка модулю складського обліку

Автор: Хоменко М. О.

Керівник проекту: Пономаренко О.В.

Нормконтролер: Кругляк В.М.

Київ 2021

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ФАХОВИЙ КОЛЕДЖ ІНЖЕНЕРІЇ ТА УПРАВЛІННЯ
НАЦІОНАЛЬНОГО АВІАЦІЙНОГО УНІВЕРСИТЕТУ»

Циклова комісія: Інженері програмного забезпечення

Освітнього ступеня: «Бакалавр»

Спеціальність: 123 Комп'ютерна інженерія

Освітньо-професійна програма: «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ

Голова циклової комісії

_____ Н.А. Рябчук

«____»_____ 2021 р.

ЗАВДАННЯ

на виконання дипломного проекту

Хоменко Марії Олегівні

1. Тема роботи: «Розробка модулю складського обліку»

затверджена наказом від «15» березня 2021 року № 28-Ст.

2. Термін виконання: з 12.04.2021р. по 20.06.2021р.
3. Вихідні дані: Модуль автоматизації складського обліку.

4. Зміст пояснювальної записки:

У 1 розділі проаналізувати завдання на проект та основні методи його розв'язання.

У 2 розділі навести опис основних етапів розробки.

У 3 розділі описати основні вимоги охорони праці.

КАЛЕНДАРНИЙ ПЛАН

виконання дипломної роботи

№ п/п	Етапи виконання кваліфікаційної роботи	Термін виконання етапів	Примітка
1.	Уточнення постановки задачі	12.04.2021	Виконано
2.	Аналіз літературних джерел	15.04.2021	Виконано
3.	Обґрунтування рішення	21.04.2021	Виконано
4.	Збір інформації	01.05.2021	Виконано
5.	Аналіз існуючих методів. Обґрунтування вибору мови програмування	06.05.2021	Виконано
6.	Виконання проекту	08.06.2021	Виконано
7.	Оформлення і друк пояснювальної записки	15.06.2021	Виконано
8.	Оформлення презентації	17.06.2021	Виконано
9.	Отримання рецензій	20.06.2021	Виконано
10.	Захист проекту		

Дипломник Хоменко М.О.

(підпис, дата) (П.І.Б.)

Дипломний керівник Пономаренко О.В.

(підпис, дата) (П.І.Б.)

Консультант з охорони праці Пешков І.В.

(підпис, дата) (П.І.Б.)

Зміст

Вступ

Склад є ключовим елементом логістичної системи. Зберігаючи витратні матеріали, товари або інші блага, він є визначним структурним підрозділом підприємства. Облік всередині складу дуже важливий. Контроль над збутом і надходженням зменшує кількість службових помилок, нестиковок, крадіжок недобросовісними працівниками, дозволяє ефективно планувати постачання, що в свою чергу підвищує ефективність використання складського простору.

Грамотна організація процесу обліку важлива для ведення бізнесу – незначні помилки можуть призвести до значних збитків. Чим більше вміст складу, тим важче облік. Одне діло, коли на складі зберігається п’ятдесят одиниць, зовсім інше коли це значення перевищує десятки тисяч. Обсяг даних складу такої потужності дуже великого об’єму, а його обробка людськими ресурсами малоефективна і має великі ризики допущення помилок.

В наш час бізнес йде пліч-о-пліч з інформаційними технологіями. В сучасних умовах розвитку економіки, виникає потреба постійного вдосконалення поточної діяльності підприємства шляхом автоматизації бізнес-процесів. Залучення інформаційних технологій до складського обліку дозволить надійно зберігати дані, зробить їх відображення більш зручним і легким в сприйняті, прискорить пошук, суттєво збільшить швидкість обробки і точність її результатів.

На сьогодні існує велика кількість програмного забезпечення для управління бізнес-процесами, що також включають в себе функції складського обліку. В більшості випадків це громіздке програмне забезпечення, яке охоплює майже всі аспекти бізнесу, що не є зручним коли потрібен конкретний модуль.

Метою проекту є створення невибагливого до технічних ресурсів модулю, що виконує функції складського обліку, і який може сумісно працювати з іншими системами або модулями підприємства.

1 Загальна частина

1.1 Постановка задачі

Метою проекту є розробка модуля, дружнього до інших систем автоматизації бізнес-процесів, що виконує функції складського обліку збільшуючи ефективність роботи його працівників і складу в цілому.

Провівши ряд досліджень предметної області, було виділено три основні задачі складського обліку:

- постачання;
- списання;
- інвентаризація.

Перші дві зазначені вище задачі ґрунтуються на арифметичних операціях над кількісними значеннями, тому потребують максимальної точності. При постачанні відбувається збільшення кількості на складі на зазначену величину, а при списанні напроти – зменшення.

Автоматизація передбачає виконання всіх обчислень системою. Корегуючи значення відповідно до результатів, модуль відображатиме актуальну кількість вантажу, який зберігається на складі.

Нажаль, повністю автоматичним складський облік зробити неможливо (система не може самостійно визначати який вантаж і скільки його прибуло, скільки чого було списано тощо), тому модуль потребує оперування користувачем.

Всього можна виділити два типи користувачів:

- адміністратор, якій відповідає за наповнення системи даними;
- оператор, який виконує функції складського обліку.

Інвентаризація є важливим механізмом складського обліку, суть якого полягає в періодичному зведенні кількісних значень обліку до фактичних. Інвентаризація відображає всі розбіжності. Допускається, коли похибка мала, но велике значення свідчить про некомпетентність працівників складу.

Облік полягає не тільки в визначенні наявного на складі вантажу, а й нагляді за тим хто і як їм оперує. При реалізації журналів, всі записи операцій над вантажем мають бути прив’язані до користувача. Завдяки цьому буде легко виявити винуватця, і покарати відповідно до уставу.

Оскільки модуль керує важливою для ведення бізнесу інформацією, важливим є забезпечення безпеки даних і попередження помилкових дій користувача, які можуть призвести до тяжких наслідків.

Для реалізації модулю була обрана клієнт-серверна архітектура, яка домінує сьогодні в сфері розробки програмного забезпечення.

Аналізуючи цю інформацію, випливає наступний ряд задач, які мають бути вирішені у проекті:

- розробка структур даних, що відображають сутності складського обліку;
- розробка надійних алгоритмів обчислення кількісних значень;
- проектування журналів складських операцій;
- розробка CRUD-запитів (create, read, update, delete) до серверу;
- розробка зручного і привабливого клієнта;
- розмежування клієнта за правами доступу;
- реалізація авторизації в системі;
- реалізація імпорту та експорту даних, для взаємодії з іншими системами;
- реалізація пошуку.

Мінімальні вимоги до системного та апаратного забезпечення для комфортної роботи модуля:

- операційна система Windows 8 (і вище), Linux;
- браузер (Chrome, Opera, Mozilla Firefox, Microsoft Edge);
- процесор Intel Core Duo;
- 2 гб оперативної пам’яті;
- 68 мб відеопам’яті;
- Стабільне підключення до серверу.

1.2 Теоретичні відомості

1.2.1 Вибір архітектурного шаблону

Архітектурні шаблони – це шаблони програмного забезпечення, що являють собою звіт «належних практик» вирішення архітектурних проблем розробки програмного забезпечення. Архітектурні шаблони виражають фундаментальну схему структурної організації певної програмної системи. Така схема складається із визначених наперед підсистем, а також точно визначає їхні сфери відповідальності та взаємовідносини.

Сервер – комп’ютер (або програма), що надає деякі послуги іншим програмам (клієнтам).

Клієнт – комп'ютер (або програма), що використовує ресурси, надані іншим комп'ютером (або програмою), який називається сервером.

Зв'язок між клієнтом і сервером зазвичай здійснюється за допомогою передачі повідомлень, часто через мережу, і використовує певний протокол для кодування запитів клієнта і відповідей сервера. Серверні програми можуть бути встановлені як на серверному, так і на персональному комп'ютері, щоразу вони забезпечують виконання певних служб (наприклад, сервер баз даних чи веб-сервер).

Залежно від розподілу функцій між клієнтом і сервером виділяються кілька типів архітектури:

- файл-сервер;
- клієнт-сервер;
- триланкова.

Файл-серверна архітектура (Рисунок 1.1) передбачає знаходження системи управління базою даних (СУБД) на клієнтській частині. Для обробки завдання користувача з сервера запитуються файли з даними, а їх обробка проводиться локально. В результаті в таких системах доводиться передавати по мережі багато даних для того, щоб на стороні клієнта серед них знайти ті, що відповідають запиту.

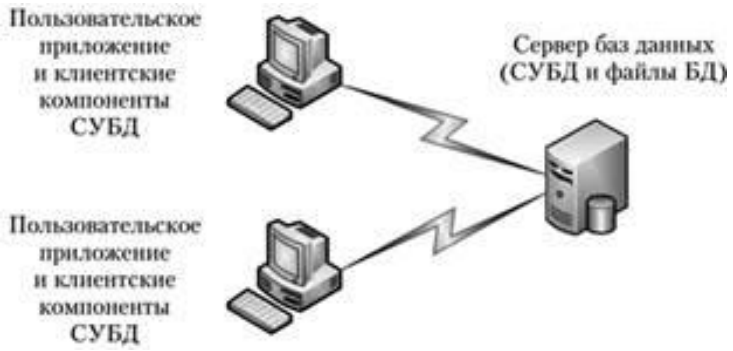

Рисунок 1.1 – Файл-сервер

Клієнт-сервер припускає, що СУБД знаходиться на сервері і тільки вона має доступ до файлів бази даних (БД). На клієнтських комп'ютерах працюють користувальницькі додатки і клієнтські компоненти, які здійснюють взаємодію з сервером. Від клієнта на сервер приходять запити, які обробляються СУБД, і результат відправляється на клієнтський комп'ютер.

Недоліком клієнт-серверу є те, що при великої кількості і географічної віддаленості клієнтів виникають проблеми з оновленням даних. Сервер підтримує відкриті з'єднання з усіма активними клієнтами, навіть якщо ніякої роботи немає. При великому числі клієнтів це може негативно впливати на продуктивність сервера БД.

Триланкова архітектура (Рисунок 1.2) є свого роду модернізацією клієнт-сервера. Вона припускає наявність додаткового сервера додатків, який проводить попередню обробку запитів клієнтів, формує запити до сервера БД і обробляє отримані результати перед відправкою їх клієнту. У триланковій архітектурі велика частина логіки додатка перенесена з клієнта на сервер, і завдання клієнтського додатка зводяться, в основному, до реалізації призначеного для користувача інтерфейсу і поданням результатів.

Рисунок 1.2 – Триланкова архітектура

Використання модулю передбачено у межах одного складу, а це означає що територіально клієнти будуть розташовані у відносно невеликому радіусі, а їх кількість не перевищуватиме декілька тисяч. Робота з копіями БД принесе багато проблем з синхронізацією спільної роботи клієнтів, велике навантаження на мережу передачі даних.

Із-за низького навантаження на сервер клієнтами, не виникає потреби залучення додаткового сервера, а вибір файл-серверної архітектури принесе лише багато незручностей. Тому було вирішено вибрати клієнт-серверну архітектуру, яка в повній мірі задовольняє потреби проекту.

1.2.2 Користувацький інтерфейс

Інтерфейс користувача – це сукупність засобів, за допомогою яких користувач спілкується з різними пристроями (з комп'ютером або побутовою технікою) або іншим складним інструментарієм (системою).

Найбільш використовуваними видами інтерфейсів є:

– графічний інтерфейс, де взаємодія з користувачем відбувається завдяки графічних об'єктів;

– інтерфейс командного рядку, де керування системою відбувається завдяки введенням користувачем раніше визначених команд.

Інтерфейс командного рядку (в народі «консоль», Рисунок 1.3) є дуже складним в засвоєні, так як потребує попереднього вивчення команд, відображати дані в ньому доволі незручно, а введення команд займає більше часу, чим скажем, натискання кнопки.

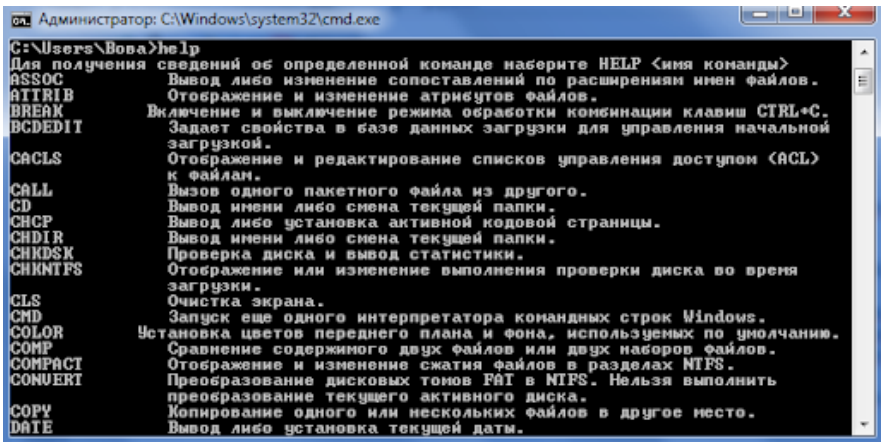

Рисунок 1.3 – Приклад інтерфейсу консолі

Графічний інтерфейс (Рисунок 1.4) є лідером завдяки своїй дружності до користувача. Графічні елементи роблять його більш привабливим і інтуїтивно зрозумілим.

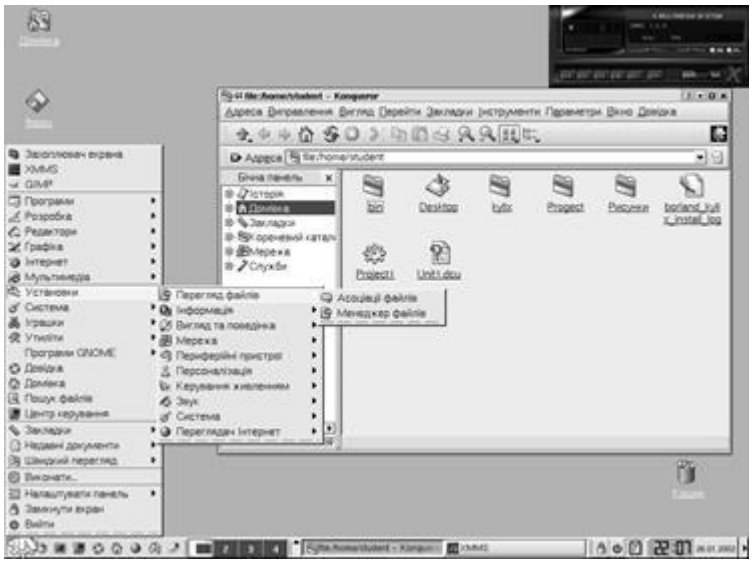

Рисунок 1.4 – Приклад графічного інтерфейсу

В веб-додатках графічний інтерфейс реалізується за допомогою html та css.

HTML – це мова тегів, засобами якої здійснюється розмічання вебсторінок для мережі Інтернет. Браузери отримують HTML-документи з вебсервера або з локальної пам'яті й передають документи в мультимедійні вебсторінки.

CSS – це спеціальна мова стилю сторінок, що використовується для опису їхнього зовнішнього вигляду.

Є багато інструментів, що включають в себе раніше описані css-класи, комбінуючи які можна швидко розробляти інтерфейс. Використання таких інструментів значно прискорює процес розробки додатку.

1.2.3 Організація збереження даних

Склад має великий об’єм інформації: матеріали, журнали поставок, журнали інвентаризацій, журнали списань тощо. Постає питання де все це зберігати.

Сьогодні популярними методами реалізації збереження даних в веб-додатках є використання json-файлів і баз даних.

JSON (Рисунок 1.5) – текстовий формат обміну даними між комп'ютерами. JSON базується на тексті, може бути прочитаним людиною. Формат дає змогу описувати об'єкти та інші структури даних. Цей формат використовується переважно для передачі структурованої інформації через мережу.

JSON будується на двох структурах:

– набір пар назва-значення (у різних мовах це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список із ключем або асоціативним масивом);

– впорядкований список значень (у багатьох мовах це реалізовано як масив, вектор, список або послідовність).


```
{
 "scores": [
   {
     "Away_Score": 2,
     "Away_Team": "Newcastle",
     "Home_Score": 2,
     "Home_Team": "Arsenal"
   },
   {
     "Away_Score": 2,
     "Away_Team": "Napoli",
     "Home_Score": 4,
     "Home_Team": "Liverpool"
   }
 ]
}
```


Рисунок 1.5 – Приклад JSON

Використання методу json-файлів корисно, коли необхідно зберігати мало зв’язні структури, но коли річ йде про збереження структур, які тісно пов’язані одна між одною, використання json стає неефективним.

База даних – це інтегрована сукупність структурованих і взаємозалежних даних, організована за певними правилами, які передбачають загальні принципи опису, зберігання і обробки даних.

Реляційна база даних – це тип бази даних, що зберігає інформацію в електронних таблицях і здійснює пошук даних в одній таблиці на підставі визначених ключових полів іншої таблиці.

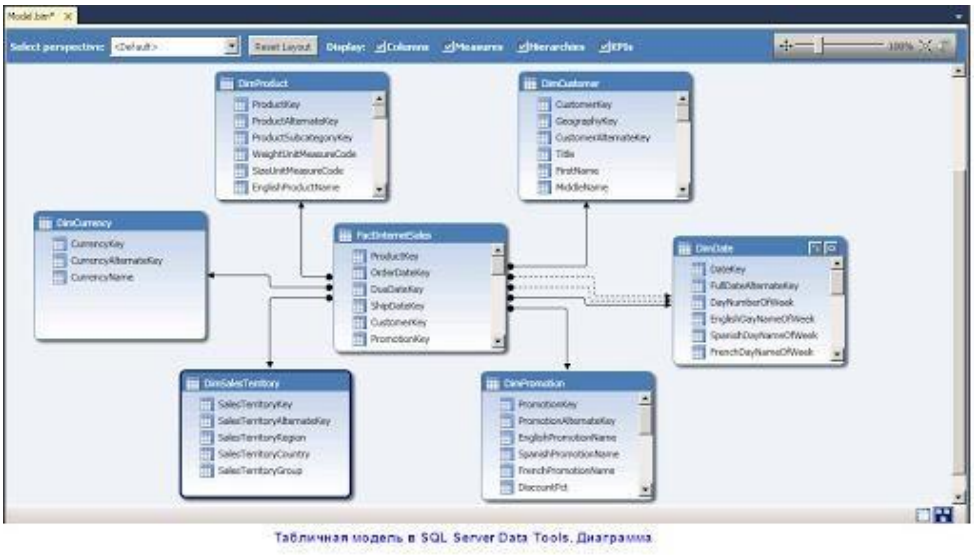

Рисунок 1.6 – Схема бази даних

В складському обліку всі журнали пов’язані з матеріалом і користувачем, який зробив внесення запису в нього, що робить використання json недоцільним.

Використання реляційної бази даних вирішить проблему зв’язності між структурами даних завдяки зовнішнім ключам, а використання оператора unique забезпечить відсутність дублювання в БД тих же матеріалів.

1.2.4 Безпека даних

Інформаційна безпека – це стан захищеності систем обробки і зберігання даних, при якому забезпечено конфіденційність, доступність і цілісність інформації, використання й розвиток в інтересах громадян або комплекс заходів, спрямованих на забезпечення захищеності інформації особи, суспільства і держави від несанкціонованого доступу, використання, оприлюднення, руйнування, внесення змін, ознайомлення, перевірки запису чи знищення.

Коли система оперує важливими для діяльності підприємства даними, вона має відповідати за безпеку. Неприпустимо щоб людина, яка не має прав доступу, впливала на корпоративну інформацію.

Розмежування інтерфейсу за правами забезпечить доступ до функцій модулю лише вповноваженим особам, але залишається питання, як ідентифікувати працівника складу.

Авторизація – це керування рівнями та засобами доступу до певного захищеного ресурсу.

Впровадження облікових записів працівників і авторизації за допомогою ім’я користувача і паролю в системі захистить дані від зловмисників.

1.2.5 Забезпечення взаємодії з іншими модулями

Модульність – принцип розробки програмної системи, що припускає реалізацію її у вигляді окремих частин (модулів).

Забезпечення взаємодії з іншими модулями структурних підрозділів є обов’язковим. На основі сформованих обліком даних відділ логістики виконує планування поставок, бухгалтерія здійснює фінансові розрахунки, керівник слідкує за діяльністю підприємства тощо. Постає питання як налагодити обмін інформацією з іншими модулями підприємства.

Запит – це спеціальний об’єкт, призначений для вибірки даних з БД, а також для виконання обчислень та інших операцій з базовими таблицями, включаючи їхнє перетворення.

Спроба звернення до сховища даних іншого модулю ненадійна. Структура даних може не відповідати очікуваній, що в кращому випадку призведе до некоректного відображення інформації, в гіршому може посприяти хибним результатам або навіть критичній помилці.

Формат файлу – це усталений стандарт запису інформації у файлі даного типу.

Створення власного формату потребує реалізації функцій парсингу (збір і систематизація інформації) файлу на іншому боці, а це значно зменшує гнучкість використання модулю, оскільки його залучення потребуватиме внесення змін в існуючі модулі.

CSV (від англ. comma-separated values) – вже існуючий файловий формат для представлення табличних даних, у якому поля відокремлюються символом коми та переходу на новий рядок.

```
1997,Ford,E350,"ac, abs, moon",3000.00
1999,Chevy,"Venture ""Extended Edition""",,4900.00
1996,Jeep,Grand Cherokee,"MUST SELL!
air, moon roof, loaded",4799.00
```

Рисунок 1.7 – Приклад вмісту файлу .csv

Формат .csv є дуже простим, тому його підтримує більшість табличних процесорів, таких як Microsoft Excel, LibreOffice та інші.

Загалом, .csv використовується не лише для зручного відображення даних у табличному вигляді, а й для перенесення даних між сховищами даних та програмними засобами.

Важко знайти мову програмування, яка не має реалізації зчитування і запису .csv файлів, що робить впровадження цього способу швидким і легким. Тому в якості проектного рішення, було обрано саме цей метод налагодження обміну даними між модулями.

1.3 Обґрунтування вибору мови програмування

Середовищем розробки було обрано Microsoft Visual Studio Code – засіб для створення, редагування та налагодження сучасних вебзастосунків і програм для хмарних систем. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X.

Редактор містить вбудований зневаджувач, інструменти для роботи з Git і засоби рефакторингу, навігації по коду, автодоповнення типових конструкцій і контекстної підказки. Продукт підтримує розробку для платформ ASP.NET і

Node.js, і позиціюється як легковагове рішення, що дозволяє обійтися без повного інтегрованого середовища розробки. Серед підтримуваних мов і технологій: JavaScript, C++, C#, TypeScript, jade, PHP, Python, XML, Batch, F#, DockerFile, Coffee Script, Java, HandleBars, R, Objective-C, PowerShell, Luna, Visual Basic, Markdown, JSON, HTML, CSS, LESS і SASS, Haxe.

Вибір мови програмування важливий етап розробки. Від правильності вибору залежить ефективність розв'язання поставлених задач. Кожна мова і технологія існує для певних завдань і цілей. Перед наданням переваги одній з них, варто спочатку визначити можливості і переваги мови.

Для реалізації клієнту модулю, була обрана мова JavaScript, а саме її фреймворк React.

Основні риси мови:

– динамічна типізація;

– автоматичне керування пам'яттю;

– прототипне програмування;

– функції як об'єкти першого класу.

Перевагами мови є:

– швидкість (JavaScript має тенденцію бути дуже швидким, оскільки він часто запускається відразу в браузері клієнта. Поки він не вимагає зовнішніх ресурсів, JavaScript не сповільнюється через виклики до серверного сервера. Крім того, всі основні браузери підтримують компіляцію JIT (вчасно) для JavaScript, що означає, що немає необхідності компілювати код перед його запуском);

– простота (синтаксис JavaScript був натхненний Java і його порівняно легко вивчити порівняно з іншими популярними мовами, такими як C ++);

– сумісність (на відміну від PHP чи інших мов сценаріїв, JavaScript можна вставити на будь-яку веб-сторінку. JavaScript можна використовувати в багатьох різних видах програм завдяки підтримці інших мов, таких як Pearl та PHP);

– навантаження на сервер (JavaScript працює на стороні клієнта, тому загалом зменшує попит на сервери);

– універсальність (є багато способів використовувати JavaScript через сервери Node.js).

JavaScript фреймворк – це інструмент, який використовується для побудови користувацьких інтерфейсів, динамічних веб-, мобільних, десктопних додатків за допомогою мови програмування JavaScript.

Фреймворки полегшують розробку продукту, оскільки пропонують функції та можливості, що вже реалізовані та можуть бути використані під час розробки продукту, і розробнику не потрібно додатково використовувати час на пошук та вирішення. Це лише одна з переваг використання фреймворків. Є також багато інших переваг:

– швидка розробка;
– код продукту стає більш структурованим і більш легшим в розумінні;
– код стає меншим;
– розробка продукту стає більш гнучкою;
– спрощує розробку продукту для декількох розробників одночасно.

Сьогодні самими популярними фреймворками є Angular, Vue та React. Фреймворки JavaScript розвиваються швидкими темпами, тому ми отримуємо часто оновлюванні версії.

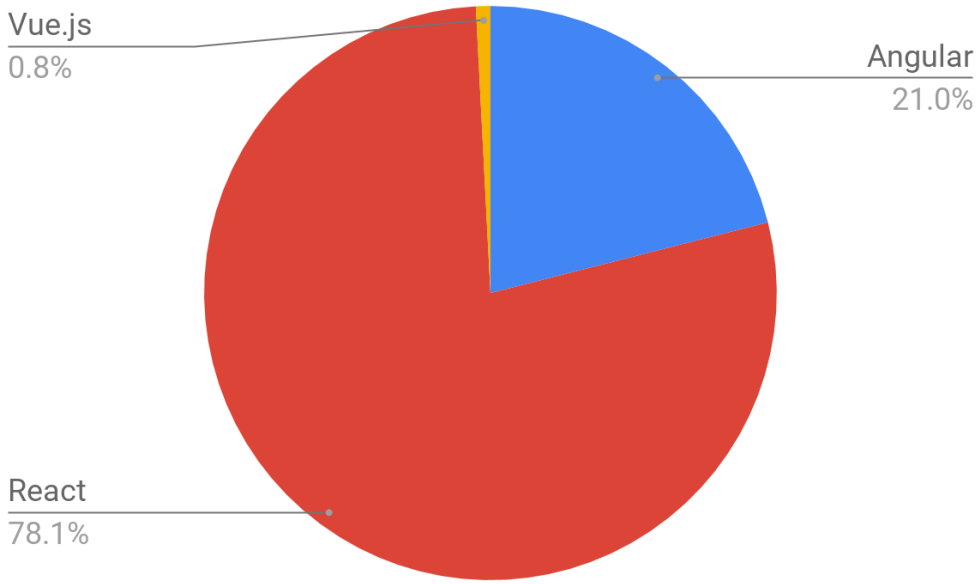

Рисунок 1.8 – Відсоток вакансій у світі, які приходяться на фреймворк

Для реалізації проекту перевага була віддана React фреймворку. Вибір був зумовлений стрімким зростанням попиту на React-розробників і рядом наступних переваг:

– легкість в навчанні (React відрізняється простотою свого синтаксису);

– високий рівень гнучкості;

– віртуальна DOM (document object model), котра дозволяє упорядкувати документи форматів HTML, XHTML чи XML в дерево, котре краще всього підходить веб-браузерам для аналізу різних елементів веб-додатків;

– поєднання з ES6/7, React може легко працювати при високому навантаженні.

2 Спеціальна частина

2.1 Опис алгоритму створення програмного засобу

Аналіз вимог полягає в визначенні потреб та умов розроблюваного продукту. Він є критичним для успішної розробки проекту. Вимоги мають бути задокументованими, вимірними, тестовими, пов'язаними з бізнес-потребами, і описаними з рівнем деталізації достатнім для конструювання системи.

На основі аналізу предметної області і визначення потреб користувачів системи, були визначені наступні вимоги до розроблюваного модулю:

- точність обчислень;
- зручний та привабливий інтерфейс;
- забезпечення безпеки даних;
- передбачення помилкових дій користувача;
- система сповіщення;
- можливість імпорту і експорту даних.

Проектування програмного забезпечення – це процес вирішення задач та планування для створення програмного рішення.

Перед початком проектування, варто виділити задачі, які має виконувати програмний модуль:

- збереження даних;
- внесення даних;
- редагування даних;
- пошук даних;
- обчислення даних;
- видалення даних.

Всі задачі оперують даними, тому для їх вирішення варто розуміти сутності складського обліку. Для кращого розуміння, була побудована наглядна схема бази даних (Рисунок 2.1).

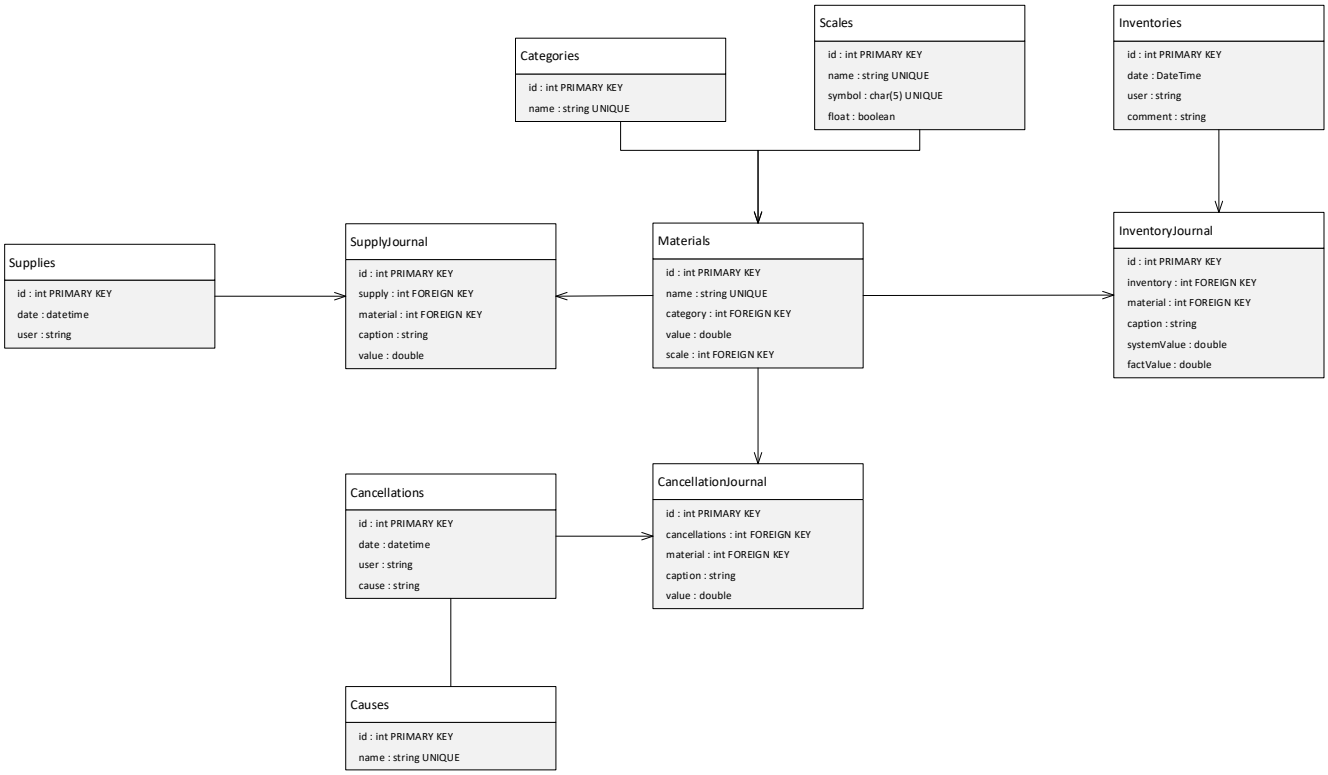

Рисунок 2.1 – Схема БД

Користувачами системи є:

– адміністратор (займається наповненням даних);

– оператор (працівник, який виконує функції складського обліку).

Варіанти використання це – опис послідовності дій, які може здійснювати система у відповідь на зовнішні дії користувачів або інших програмних систем. Варіанти використання відображають функціональність системи.

Метою розробки діаграм прецедентів є формування загальних вимог до функціональної поведінки проектованої системи, розробка початкової концептуальної моделі системи, її подальшої деталізації у формі логічних і фізичних моделей, підготовка початкової документації для взаємодії розробників системи з замовниками і користувачами.

Наступні діаграми варіантів використання були побудовані з метою відображення функцій модулю, з якими можуть взаємодіяти оператор (Рисунок 2.2) і адміністратор (Рисунок 2.3).

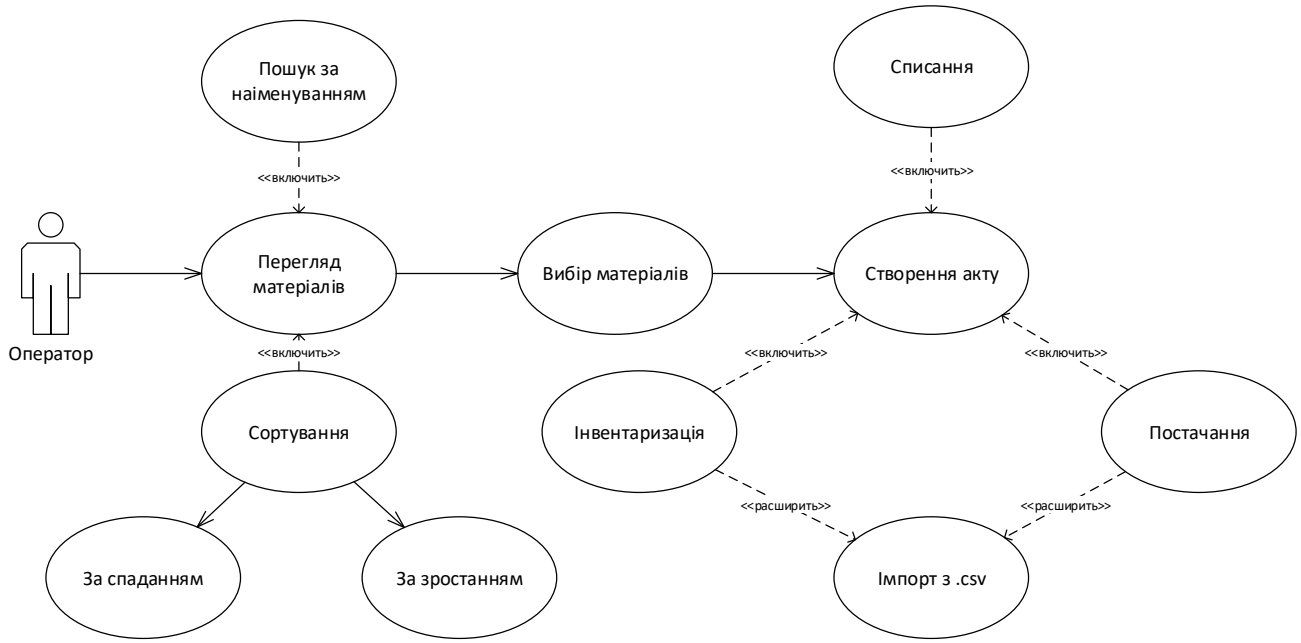

Рисунок 2.2 – Варіанти використання оператора

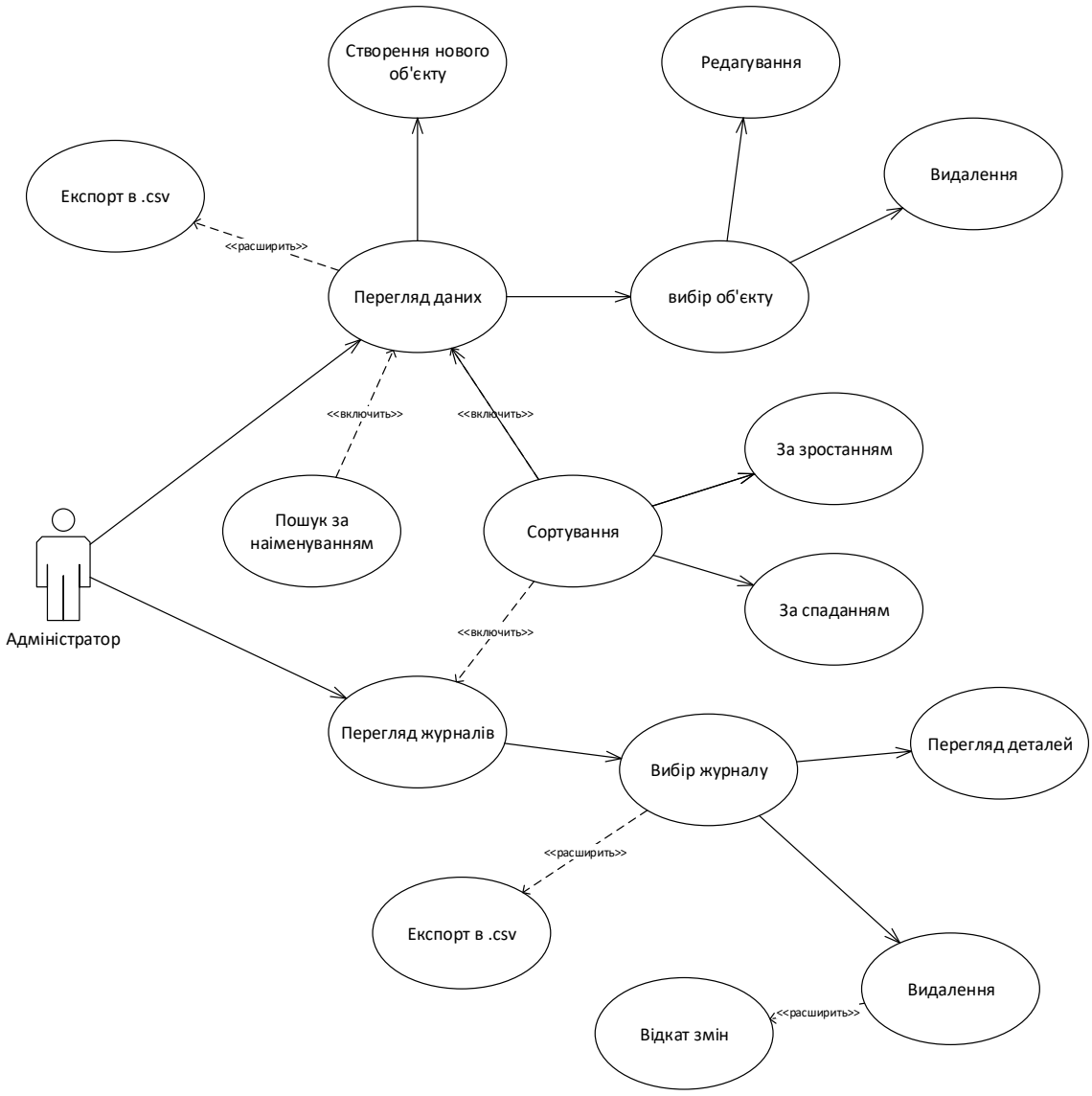

Рисунок 2.3 – Варіанти використання адміністратора

На діаграмах дані і журнали зображенні як узагальнення, оскільки над структурами які в них входять виконуються одні і ті самі функції.

Під даними розуміються:

– категорії;

– одиниці;

– матеріали.

Журнали включають в себе три основні задачі обліку: інвентаризації, постачання і списання.

Початковою точкою роботи є авторизація в системі. Після її виконання, починається життєвий цикл клієнта (Рисунок 2.4), який продовжується до виходу з облікового запису чи закриття програми.

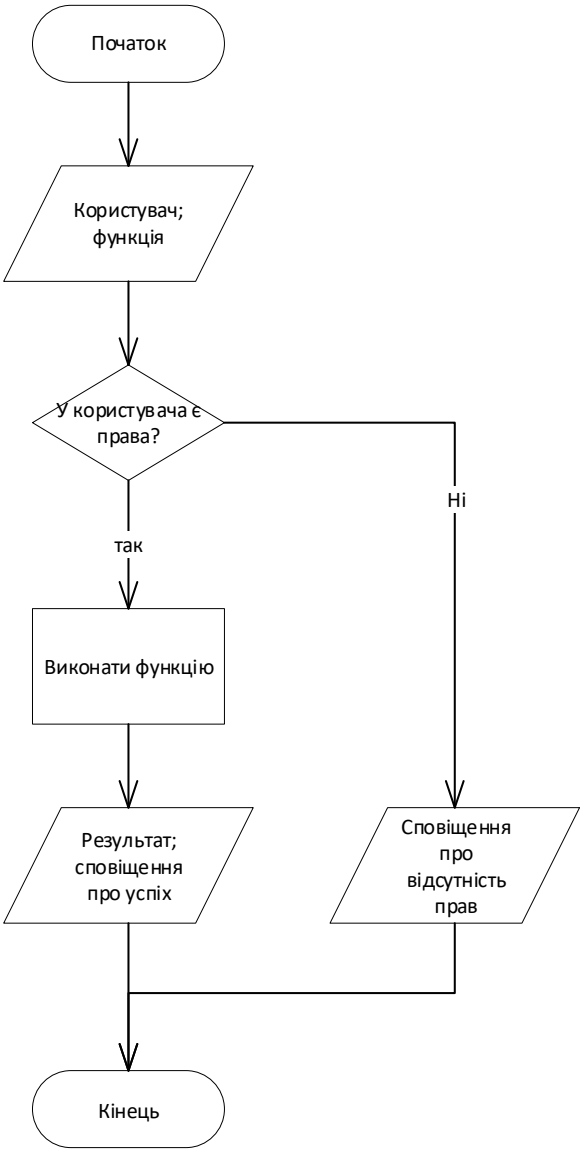

Рисунок 2.4 – Алгоритм взаємодії користувача з модулем

Діаграми послідовності є одним із способів формалізації сценаріїв використання. Її перевага закладається в тому, що на ранніх стадіях опису сценаріїв можливо з’ясувати склад взаємодіючих компонентів та описати потік повідомлень від одних компонентів до інших. Ці компоненти та потоки повідомлень в подальшому будуть трансформовані в конкретні класи (об’єкти), методи цих об’єктів.

Побудована діаграма послідовності (Рисунок 2.5) відображає взаємодію між користувачем модуля, клієнтською і серверною частинами. Для демонстрації було обрано функцію внесення даних.

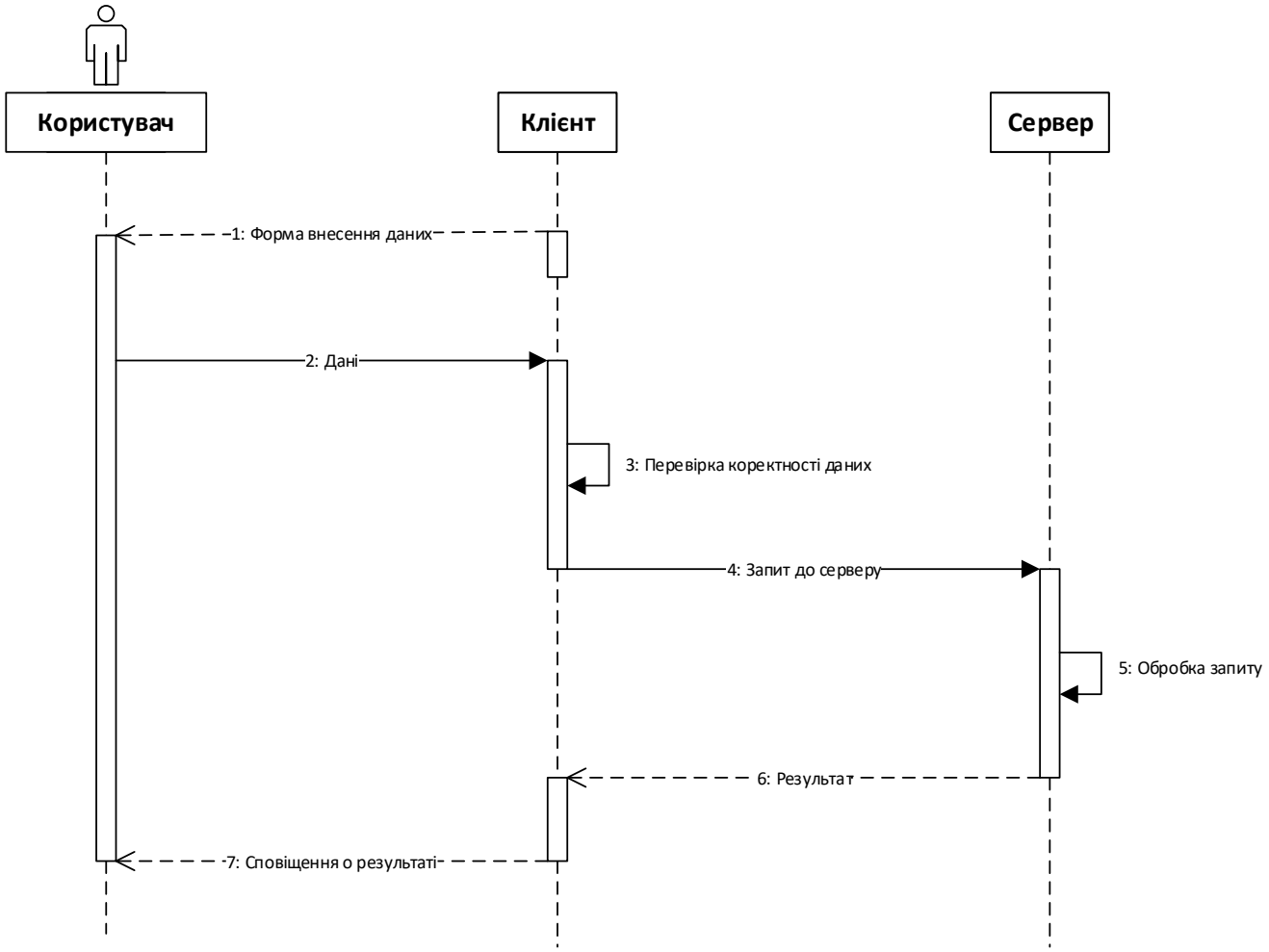

Рисунок 2.5 – Діаграма послідовності

2.2 Опис засобів реалізації

2.2.1 Кастомізація

Для кастомізації компонентів і сторінок у цілому була обрана утиліта Tailwind, яка містить в собі css-класи, комбінуючи які можна швидко компонувати об’єкти на сторінки і придавати їм привабливий зовнішній вигляд.

Tailwind має наступний ряд переваг:

– має малий вплив на продуктивність (він автоматично видаляє всі непотрібні в проекті стилі);

– адаптивність (спрощує процес адаптивної верстки, оскільки має вже раніше написані медіа-запити);

– реалізує стани фокусу і наведення.

```
className={"rounded-full text-sm text-white font-bold py-3 px-5 " + color}
```

Рисунок 2.6 – Приклад кастомізації

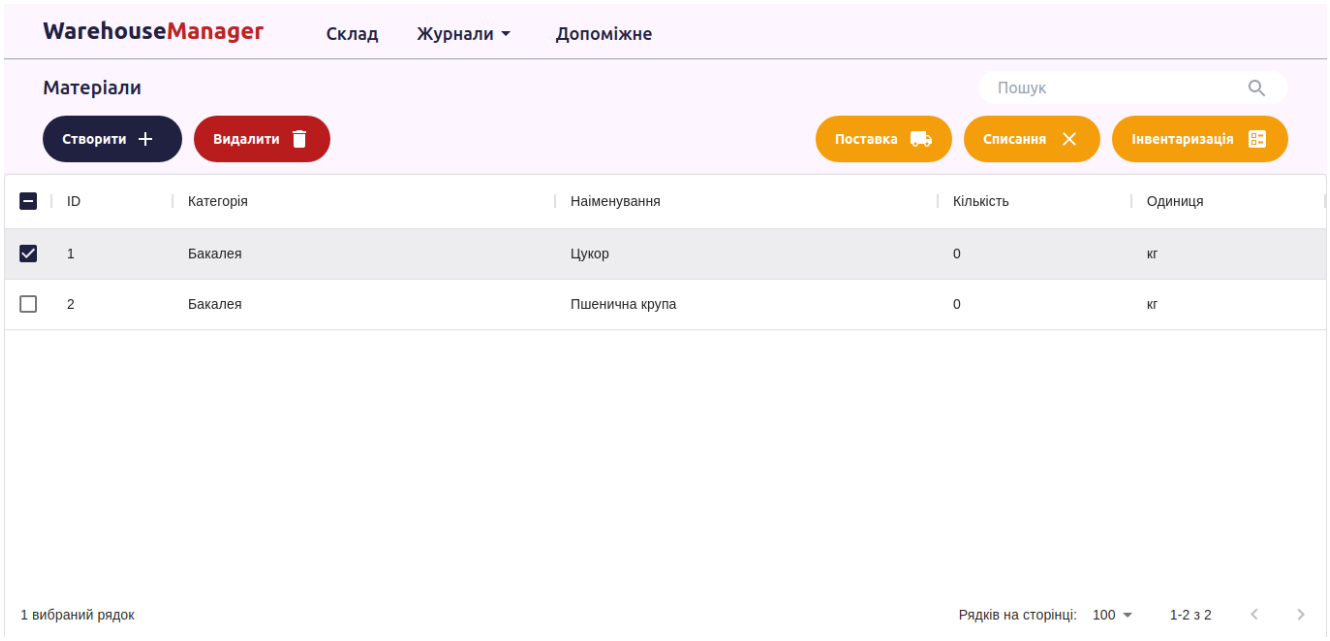

Рисунок 2.7 – Зовнішній вигляд модулю

2.2.2 Сутності

Модуль виконано в функціональному стилі. Об’єкти сутностей зберігаються в масивах і передаються через контексти в ієрархії компонентів.

Головною сутністю модулю є Material (матеріал) – вона відображає вміст складу.

Матеріал містить наступні поля:

- name (найменування);
- category (категорія матеріалу);
- value (кількість матеріалу на складі);
- scale (одиниця виміру матеріалу).

Допоміжними сутностями є Category (категорія), Scale (одиниця), Cause (підстава).

Категорія і підстава є простими структурами, які мають єдине поле – name. Категорія створена для зручного групування матеріалів за спільними ознаками. Підстава створена з метою налаштування керівником основ, на яких оператори можуть списувати матеріали (наприклад, «вийшов строк придатності» чи «переміщення між складами»).

Одиниця виміру описує в яких величинах вимірюється матеріал. Завдяки неї модуль підстроюється під складську систему. Вона має наступні поля:

- name (найменування);
- symbol (скорочення найменування, наприклад кілограм – кг);
- float (визначає чи ціла величина чи ні).

Inventory (інвентаризація), Supply (поставка), Cancellation (списання) описують основні задачі складського обліку. Вони мають спільні поля:

- user (користувач, який створив);
- created_at (дата і час, коли було створено).

Списання містить поле cause (підстава), яка описує причину списання матеріалу зі складу.

Інвентаризація, зокрема періодичного зведення кількості в системі до фактичної, виконує функцію редагування. Це рішення було прийнято, оскільки просте редагування не залишає записів в системі, і це може сприяти махінаціям на складі. Інвентаризація містить в собі системне і фактичне значення показуючи які були внесені зміни. Обов'язковим полем інвентаризації є comment, в яке вписується причина інвентаризації, будь то планова чи редагування даних по причині введення помилкових даних чи тому подібне.

Ті же сутності складських задач з приставкою Journal, містять в собі записи відповідних задач, тобто несуть собі пару матеріал-кількість.

В журналах поля, які пов'язані з іншими таблицями вносяться строковим значенням. В першу чергу причиною є те, що при видаленні сутності, порушуються зв'язки між таблицями. До того ж для звітності важливим є те, яким був матеріал на той момент часу, а не зараз (у випадку редагування найменування). Це також запобігає махінаціям.

Зв'язок з матеріалом підтримується для зручного виведення історії матеріалу, але відображеним полем буде caption, який зберігає в собі найменування, яке було на момент створення задачі.

2.2.3 Сервер

Взаємодія клієнта з сервером реалізована за допомогою Axios.

Axios – це дуже популярна бібліотека JavaScript, яку можна використовувати для виконання запитів HTTP. Він працює як на платформах Browser, так і на Node.js.

Дані між сторінками передаються за допомогою контекстів, які постачають провайдери.

Реалізація спілкування з сервером розділена на три компоненти:

- api (містить реалізацію запитів до серверу);
- hooks (містить виклик запитів і їх синхронізацію з даними клієнта);
- providers (описує провайдер, що постачає дані з сервера).

Кожен компонент звертається до компоненту вище, таким чином утворюючи ієрархічну структуру.

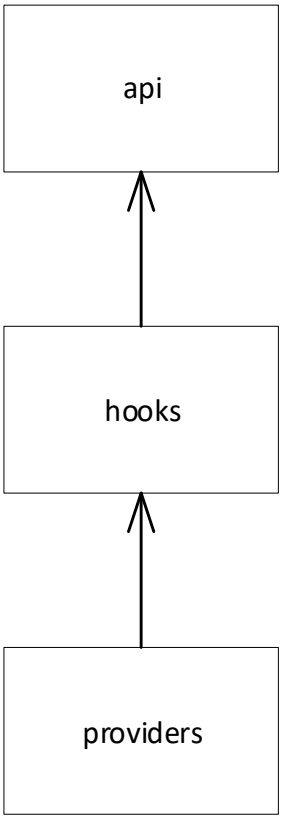

Рисунок 2.8 – Структура спілкування з сервером

Розглянемо на прикладі матеріалів.

Api:

```
export const getMaterials = async (query = "") =>
(await axios.get(`materials${query}`)).data;
export const getMaterial = async (id) =>
(await axios.get(`materials/${id}`)).data;
export const getMaterialsCount = async () =>
  (await axios.get("materials/count")).data;
export const postMaterial = async (material) =>
  (await axios.post("materials", { ...material })).data;
export const putMaterial = async ({ id, ...material }) =>
  (await axios.put(`materials/${id}`, material)).data;
export const deleteMaterial = async (id) => {
  return (await axios.delete(`materials/${id}`)).data;};
```

Hook:

```
import { propEq, reject } from "ramda";
import { useCallback, useState } from "react";
import {
  deleteMaterial,
  getMaterials,
  postMaterial,
  putMaterial,
} from "../api/material";
import { alter } from "../helpers/array";
const useMaterials = () => {
  const [materials, setMaterials] = useState([]);
  const [selected, setSelected] = useState([1, 2]);
  return {
    materials,
    setMaterials,
    selected,
    setSelected,
    retrieve: useCallback(
      async (query = "") => setMaterials(await getMaterials(query)), []),
    remove: useCallback(
      async (id) => {
        await deleteMaterial(id);
        setMaterials([...reject(propEq("id", id), materials)]);
      }, [materials]),
    create: useCallback(
      async (data) => {
        const material = await postMaterial(data);
        setMaterials([...materials, { ...material }]);
        return material;
      }, [materials]),
    update: useCallback(
      async (payload) => {
        const material = await putMaterial(payload);
        setMaterials(
```

```
        alter(propEq("id", material.id), materials, (m) => ({
          ...m,
          ...material,
        }))
      );
      return material;
    }, [materials]),
  };
};
export default useMaterials;
```

Provider:

```
import { createContext, useContext } from "react";
import useMaterials from "../../hooks/useMaterials";
export const MaterialsContext = createContext({});
export const useMaterialsContext = () => useContext(MaterialsContext);
const MaterialsProvider = ({ children }) => (
  <MaterialsContext.Provider value={useMaterials()}>
    {children}
  </MaterialsContext.Provider>
);
export default MaterialsProvider;
```

2.2.4 Відображення даних

Відображення даних відбувається в табличному вигляді. Для реалізації був запозичений компонент з бібліотеки Material UI – DataGrid. Його логіка однакова для всіх сутностей, тому щоб постійно не описувати його на кожній сторінці (що значно ускладнить його редагування), його було винесено в окремий файл (DataView) і відкориговано відносно потреб проекту.

```
import { useState, useEffect } from "react";
import { Paper } from "@material-ui/core";
import { DataGrid } from "@material-ui/data-grid";
export default function DataView({
```

```
  columns = [],
  rows = [],
  target = null,
  setTarget = () => null,
  setSelected = () => null,
  TransferToView = (array = []) => {
    return array;
  }, ...props
}) {
  const [view, setView] = useState([]);
  useEffect(() => {
    setView([...TransferToView(rows)]);
  }, [rows]);
  return (
    <Paper className="h-full w-full max-h-full max-w-full">
      <DataGrid
        columns={columns}
        rows={view}
        rowsPerPageOptions={[10, 25, 50, 100]}
        onRowClick={(e) => {
          if (target === e.id) setTarget(null);
          else setTarget(e.id);
        }}
        onSelectionModelChange={(e) => {
          setSelected(e.selectionModel);
        }}
        disableSelectionOnClick
        {...props}
      />
    </Paper>
  );
}
```

Вхідними даними компоненту є columns (стовпці), rows (строки), target (обраний елемент), setTarget (встановити обраний елемент), setSelected (встановити обрані елементи), TransferToView.

Columns зберігає заголовки таблиці, в яких вказано ідентифікатор стовпця і його параметри (найменування, ширина тощо). Rows – це масив об’єктів, поля яких повинні співпадати з ідентифікаторами стовпців, інакше це поле не буде відображатись в очікуваному стовпці.

Target та setTarget являються станом і його сеттером, що зберігає в собі обраний натисканням на строку об’єкт. setSelected, на відміну від попередника, встановлює в стан selected масив обраних checkbox-ом об’єктів. Target використовується при редагуванні чи видаленні, в той час як selected в множинному виборі матеріалів для складської задачі.

При відображені складних структур є проблема – с серверу приходить наступна відповідь (матеріал):

```
{
        id : some_id,
        name: some_name
        …
        category:
        {
                …
        }
}
```

Матеріал має зв’язок з категорією і одиницею, і вони відображаються не іменами чи id, а об’єктами, із-за чого їх свиникає потреба нормалізації даних.

Функція TransfetToView переробляє структуру під потрібний для відображення варіант. Якщо ця функція не була визначена, за замовченням вона повертає rows.

View зберігає в собі нормалізовані дані і відіграє роль області відображення. useEffect(…, [rows]) передбачає що кожного разу, коли над rows будуть відбуватись якісь дії, вони будуть встановлені в view. Таким чином, користувачу будуть

відображені на екрані всі оновлення сутностей без необхідності перезавантаження сторінки.

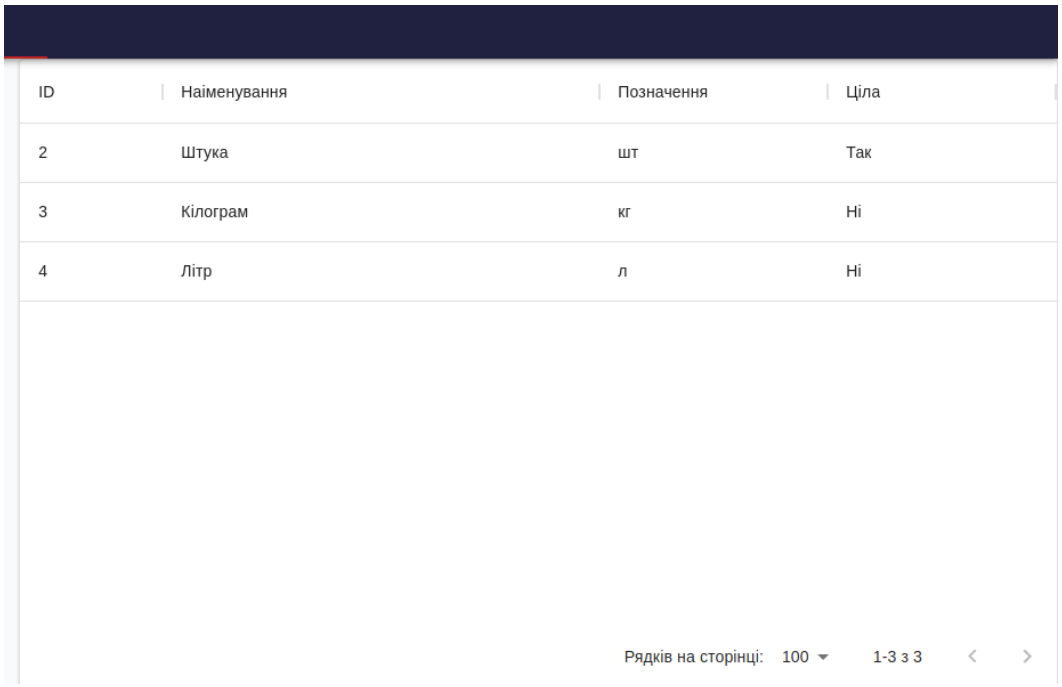

Рисунок 2.9 – DataView Scales

2.2.5 Пошук

Для пошуку був створений компонент Search, який приймає в себе масив об’єктів.

```
import SearchIcon from "@material-ui/icons/Search";
import { useState, useEffect } from "react";
export default function Search({
  array = [],
  setView = () => null,
  TransferToView = (result) => {
    return result;
  },
}) {
  const [value, setValue] = useState(null);
  useEffect(() => {
    if (value) {
```

```
    var result = [];
    array.forEach((element) => {
      if (element.name.toUpperCase().match(value.toUpperCase()))
        result.push(element);
    });
    setView([...TransferToView(result)]);
  } else setView([...TransferToView(array)]);
 }, [value]);
 return (
  <div className="flex flex-row rounded-full bg-white py-1 px-5 space-x-2">
    <input
      type="text"
      placeholder="Пошук"
      className="w-full focus:outline-none"
      onChange={(e) => {
        setValue(e.target.value);
      }}
    />
    <div className="text-gray-400">
      <SearchIcon />
    </div>
  </div>
 );
}
```

Компонент був створений в більшій мірі для матеріалів, но його можна залучити до інших сутностей, що мають поле name, оскільки пошук здійснюється саме по ньому.

Value є значенням рядку пошуку. Встановлена залежність від нього дозволяє виконувати динамічний пошук. Модуль приводить значення рядку пошуку і найменування об’єкту до спільного регістру (верхнього) і шукає співпадіння.

Якщо рядок пошуку пуста – система автоматично виведе всі об’єкти сутності.

Рисунок 2.10 – Рядок пошуку

2.2.6 Система сповіщень

Система сповіщень в модулю реалізована спливаючими повідомленнями в правому верхньому куту екрану. Повідомлення спливають по черзі однин за одним і відображаються протягом 3-х секунд.

Всього модуль передбачує наступні види повідомлень:

- success;
- warning;
- error;
- info.

Для їх постачання на сторінки був реалізований провайдер:

```
export const NotificationsContext = createContext({});
export const useNotificaitonsContext = () => useContext(NotificationsContext);
const NotificationsProvider = ({ children, ...props }) => {
  const [notifications, setNotifications] = useState([]);
  const timer = useRef(null);
  const pop = () => {
    setNotifications(remove(0, 1, notifications));
  };
  const close = (id) => {
    setNotifications(reject(propEq("id", id), notifications));
  };
  const notify = ({ props = {}, content }) => {
    const id = nanoid();
    setNotifications([
      ...notifications,
      {
        id,
        props,
        content,
      },
    ]);
  };
```

```
const success = (content, props = {}) => {
  notify({ content, props: { ...props, severity: "success" } });
};
const error = (content, props = {}) => {
  notify({ content, props: { ...props, severity: "error" } });
};
const warning = (content, props = {}) => {
  notify({ content, props: { ...props, severity: "warning" } });
};
const info = (content, props = {}) => {
  notify({ content, props: { ...props, severity: "info" } });
};
useEffect(() => {
  timer.current = clearInterval(timer.current);
  if (!isEmpty(notifications)) {
    timer.current = setInterval(pop, 3000);
  }
}, [notifications]);
return (
  <NotificationsContext.Provider
    {...props}
    value={{
      notifications,
      notify,
      success,
      warning,
      error,
      info,
      close,
    }}
  >
    <div
      className="focus:outline-none fixed right-0"
      style={{
        zIndex: 2000,
```

```
    }}
  >
    <ul className="py-6 px-6 flex flex-col  items-end">
      <TransitionGroup>
        {notifications.map(({ id, props, timeout, content }) => (
          <CSSTransition
            key={id}
            classNames={{
              enter: "opacity-0",
              enterActive: "opacity-100",
              enterDone: "opacity-100",
              exit: "opacity-100",
              exitActive: "opacity-0",
              exitDone: "opacity-0",
            }}
            timeout={300}
          >
            <li
              className="my-2.5 transition-all duration-300 transform shadow-xl bg-transparent pointer-events-auto"
              style={{
                width: "420px",
              }}
            >
              <Alert
                {...props}
                onClose={() => {
                  close(id);
                }}
              >
                {typeof content === "function" &&
                  content({
                    id,
                    props,
                    timeout,
```

```
                close: () => close(id),
              })}
            </Alert>
          </li>
        </CSSTransition>
      ))}
    </TransitionGroup>
  </ul>
 </div>
 {children}
 </NotificationsContext.Provider>
 );
};
export default NotificationsProvider;
```

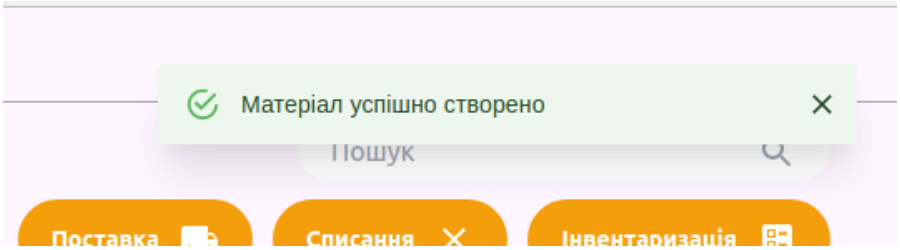

Рисунок 2.11 – Success сповіщення

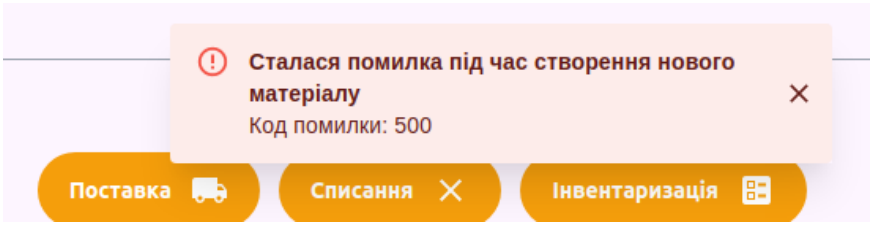

Рисунок 2.12 – Error -сповіщення

2.2.7 Створення та редагування об’єкту

Для реалізації створення і редагування об’єктів, були розроблені окремі компоненти (форми), які можна залучати до сторінок або інших компонентів. Таке проекте рішення було прийнято тому, що це дозволить вставляти ці форми в різні частини проекту, і забезпечить синхронізацію інтерфейсу. Таким чином буде дуже зручно вносити зміни в модуль.

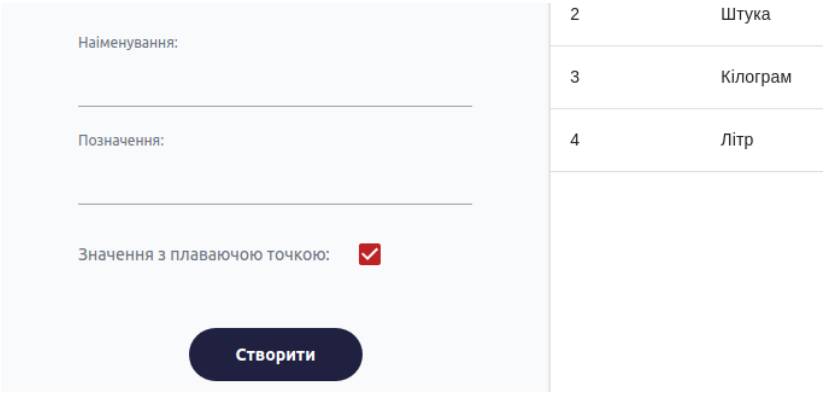

Рисунок 2.13 – Приклад вкладеної форми в панель

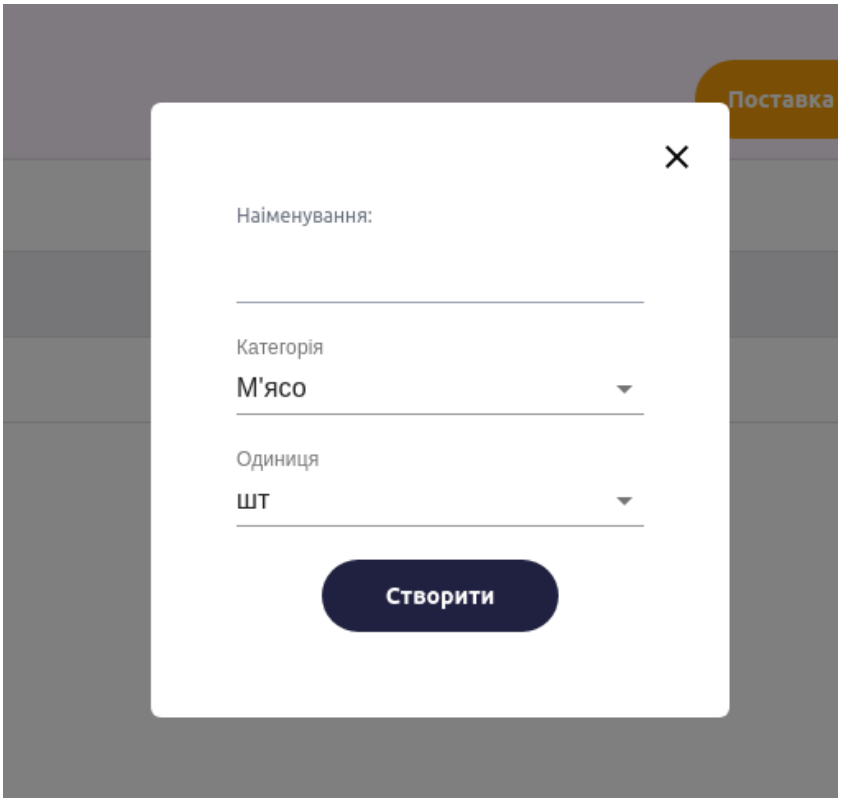

Рисунок 2.14 – Приклад вкладеної форми в модальне вікно

Форми працюють як в режимі створення, так і редагування. Вона приймає в себе target, і якщо він пустий створює новий об’єкт, інакше редагує переданий.

```
import { useCategoriesContext } from "@/src/providers/categories";
import { useNotificaitonsContext } from "@/src/providers/notifications";
import { useState, useEffect } from "react";
import { useForm } from "react-hook-form";
import { find, propEq } from "rambda";
import TextInput from "@/components/TextInput";
import Submit from "./submit-button";
export default function CreateOrUpdateCategory({
```

```
  target = null,
  setTarget = () => null,
  children,
}) {
  const {
    register,
    handleSubmit,
    setValue,
    formState: { errors },
  } = useForm();
  const { categories, create, update } = useCategoriesContext();
  const { success, error, warning } = useNotificaitonsContext();
  const [category, setCategory] = useState(null);
  useEffect(() => {
    setCategory(find(propEq("id", target), categories));
  }, [target]);
  useEffect(() => {
    setValue("id", category ? category.id : null);
    setValue("name", category ? category.name : null);
  }, [category]);

  const onSubmit = (data, e) => {
    if (category) {
      update(data)
        .then(() => {
          success(() => <p>Категорія успішно оновлена</p>);
          setTarget(null);
        })
        .catch((e) => {
          error(() => (
            <>
              <p className="font-bold">
                Сталася помилка під час оновлення категорії
              </p>
              <p>Код помилки: {e.response && e.response.status}</p>
```

```
          </>
        ));
      });
  } else {
    create(data)
      .then(() => {
        success(() => <p>Категорія успішно створена</p>);
        setTarget(null);
      })
      .catch((e) =>
        error(() => (
          <>
            <p className="font-bold">
              Сталася помилка під час створення нової категорії
            </p>
            <p>Код помилки: {e.response && e.response.status}</p>
          </>
        ))
      );
  }
  e.target.reset && e.target.reset();
};
return (
  <div>
    <form
      onSubmit={handleSubmit(onSubmit)}
      className="flex flex-col w-full space-y-5"
    >
      <div className="flex flex-row">
        <TextInput
          label="Наіменування:"
          type="text"
          defaultValue={category ? category.name : ""}
          {...register("name", { required: true })}
          onChange={(e) => {
```

```
            setValue("name", e.value);
          }}
        />
      </div>
      <div className="flex flex-row text-sm text-red-700">
        {errors.name && <span>введіть наіменування...</span>}
      </div>
      <div className="flex flex-row justify-center">
        <Submit object={target} />
      </div>
    </form>
    <div className="flex flex-col space-y-3 mt-3">{children}</div>
  </div>
 );
}
```

Target несе в собі id об’єкта. Встановлена залежність useEffect від target забезпечує постійний пошук і вибір об’єкту з масиву сутності.

Використання useForm дозволяє визначати правила полів форми, що запобігає введенню неприпустимих значень і відправки даних з ними. Register містить в собі значення полів, а errors записує помилки. Виведення errors дозволяє сповіщати користувачів про допущену в полі помилку.

При відправці форми виконується запит. Функція then визначає що при успішному завершенні будуть виконані наступні дії (у нашому випадку, модуль сповістить про успішне виконання запиту. Catch ловить помилки і виводить в повідомлення її код.

2.2.8 Видалення об’єкту

Видалення також підв’язане під target. Для запобігання помилкового видалення інформації, модуль потребує підтвердження користувача в модальному вікні.

```
import Button from "@/components/Button";
```

```
export default function DeleteForm({
  onDelete = () => null,
  onCancel = () => null,
  children,
}) {
  return (
    <div className="flex flex-col space-y-10">
      <div className="flex flex-row justify-center">
        <span>Підтверджуете видалення?</span>
      </div>
      <div className="flex flex-row justify-center space-x-5">
        <Button
          onClick={onDelete}
          color="w-24 bg-primary-main hover:bg-primary-light"
        >
          Так
        </Button>
        {children}
        <Button onClick={onCancel} color="w-24 bg-red-700 hover:bg-red-600">
          Hi
        </Button>
      </div>
    </div>
  );
}
export default function Delete({ target, setTarget = () => null }) {
  const { remove } = useCausesContext();
  const [open, setOpen] = useState(false);
  const { success, info, error } = useNotificaitonsContext();

  const onClose = () => {
    setOpen(false);
  };

  return (
```

```
    <div>
      <button
        className="w-36 px-5 py-3 rounded-full text-white text-sm font-bold bg-red-700 hover:bg-red-600"
        onClick={() => {
          if (target) setOpen(true);
          else info(() => <p>Оберіть елементи для видалення</p>);
        }}
      >
        Видалити
      </button>
      <ModalWindow open={open} onClose={onClose}>
        <DeleteForm
          onCancel={onClose}
          onDelete={() => {
            remove(target)
              .then(() => {
                success(() => <span>Підстава видалена успішно</span>);
                setTarget(null);
                onClose();
              })
              .catch((e) =>
                error(() => (
                  <>
                    <p>При видалені підстави сталась помилка</p>
                    <p>Код помилки: {e.response && e.response.status}</p>
                  </>
                ))
              );
          }}
        />
      </ModalWindow>
    </div>
  );
}
```

2.2.9 Створення задачі

Реалізація журналу для всіх трьох складський задач подібна. Вона включає наступні кроки:

- вибір матеріалів;
- встановлення кількості;
- підтвердження створення.

Процес створення відбувається ланцюгом подій, де при успішному виконанні запиту викликається наступний. Тобто спочатку створюється сама задача, якщо результат успішний, наступними будуть створюватись записи відповідних матеріалів. Останнім кроком, відповідно буде виконання обчислень і редагування кількості матеріалів в системі.

Для прикладу розглянемо задачу постачання:

```
const [data, setData] = useState([]);
  const { materials, selected, setSelected, update, retrieve } =
    useMaterialsContext();
  const { user } = useAuthorizationContext();
  const { CreateSupply: create } = useSuppliesContext();
  const { CreateRecord: create } = useSupplyJournal();
  const onCreate = () => {
    const supply = CreateSupply({ user: user })
      .then(() => {
        data.forEach((obj) => {
          CreateRecord({
            supply: supply.id,
            material: obj.material.id,
            value: obj.value,
          })
            .then(() => {
              var material = obj.material;
              material.value += obj.value;
              update(material).catch((e) => {
```

```
            error(() => (
              <>
                <p className="font-bold">
                  Сталася помилка під час оновлення значення матеріалу
                </p>
                <p>Код помилки: {e.response && e.response.status}</p>
              </>
            ));
          });
        })
        .catch((e) => {
          error(() => (
            <>
              <p className="font-bold">
                Сталася помилка під час внесення запису {obj.material.name}
              </p>
              <p>Код помилки: {e.response && e.response.status}</p>
            </>
          ));
        });
    });
  })
  .catch((e) => {
    error(() => (
      <>
        <p className="font-bold">
          Сталася помилка під час створення поставки
        </p>
        <p>Код помилки: {e.response && e.response.status}</p>
      </>
    ));
  });
};
```

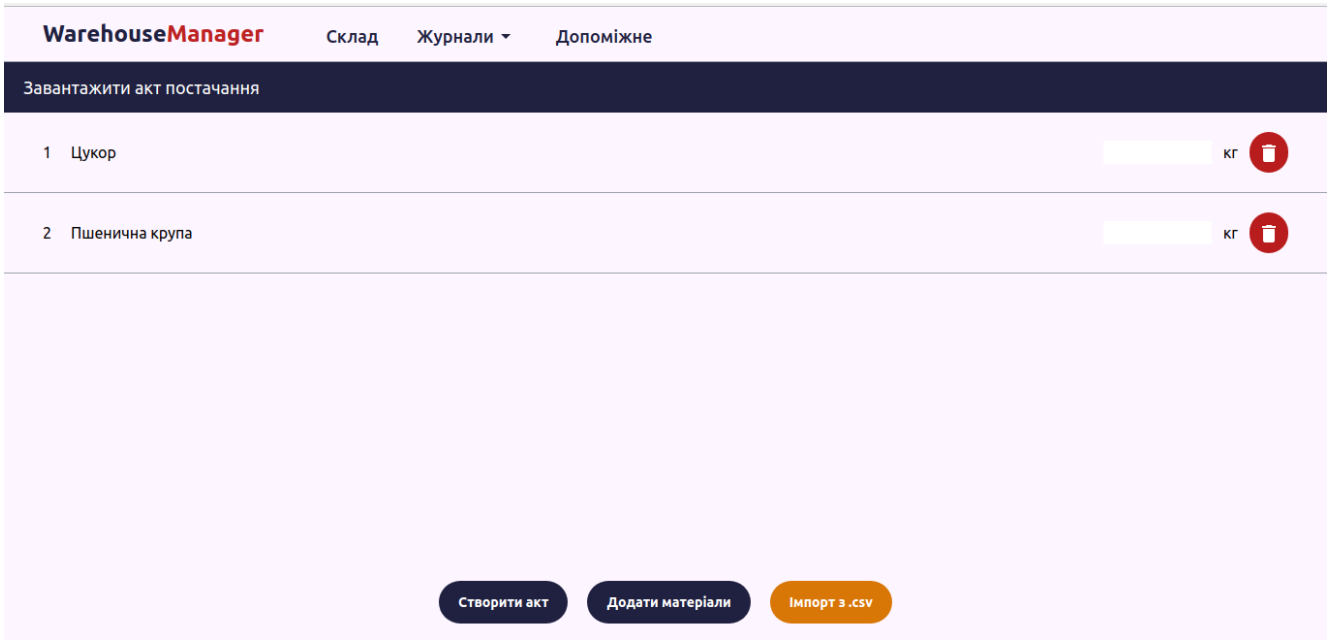

Рисунок 2.15 – Форма створення поставки

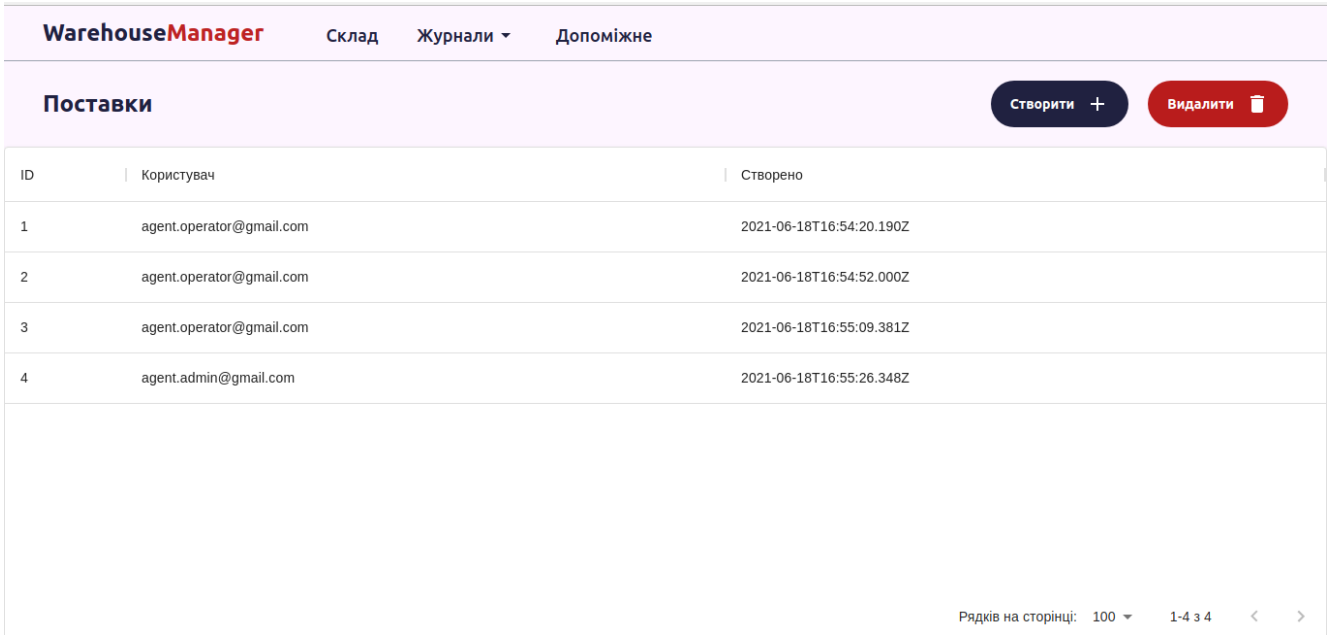

Рисунок 2.16 – Відображення поставок

2.2.10 Імпорт та експорт

Для реалізації імпорту та експорту були створені окремі компоненти, які приймають в себе path (у випадку імпорту) та path, data, header у випадку експорту.

При імпорту, компонент приймає path (шлях до файлу) і повертає data (зчитані дані). При експорту компонент приймає і шлях, і дані (відповідно файл, в

який буде записана інформація і дані, які будуть записані. Третій вхідний аргумент header описує заголовки стовпців таблиці.

Експорт:

```
export default function WriteToCsv(path, header, data) {
  const csvWriter = createCsvWriter({
    path: path,
    header: header,
  });
  csvWriter.writeRecords(data);
```

Імпорт:

```
export default function ReadFromCvs(path) {
  const fs = require("fs");
  const path = require("path");
  const csv = require("csv-parser");
  var result = [];
  fs.createReadStream(path)
    .pipe(csv())
    .on("data", (data) => result.push(data))
    .on("end", () => {
      return result;
    });
}
```

2.3 Порівняльний аналіз реалізованого програмного засобу та програм-аналогів

2.3.1 РемОнлайн

Можливості програмного засобу:

– збереження виробів клієнтів та керування їх логістикою (реєструйте в РемОнлайн всі пристрої, транспорт, техніку та інші вироби клієнтів, які вони здають вам на обслуговування. Налаштуйте автоматичне переміщення виробів між

складами відповідального зберігання, розміщуйте їх в складських осередках, щоб знати точне місце розташування, та відстежуйте історію переміщень);

– розділення товарів та витратних матеріалів з орендним майном і підмінним фондом (зберігайте власне майно, яке здаєте в оренду або видаєте клієнтам як підмінний фонд, окремо від товарів на продаж і витратних матеріалів. Фіксуйте передачу виробів клієнтам в замовленнях та налаштуйте їх автоматичне повернення на склад при закритті замовлення. Встановлюйте орієнтовну вартість для простого розрахунку застави);

– ведення обліку складських залишків (контролюйте наявність товарів на складі та відстежуйте їх залишки: переглядайте всі позиції, сортуйте товари по наявності, категоріям і складам. Імпортуйте й експортуйте товари, друкуйте цінники та етикетки. РемОнлайн забезпечує повну автоматизацію складського обліку від оприбуткування до списання товарів);

– відстеження кожної одиниці на складі (увімкніть серійний облік і відстежуйте історію пересування кожної товарної одиниці. Вказуйте існуючі серійні номери або генеруйте їх за допомогою РемОнлайн, друкуйте етикетки зі штрихкодами при оприбутковуванні. А функція адресного зберігання в програмі складського обліку дозволяє створювати на складі в РемОнлайн комірки для розміщення. Завдяки програмі ви знатимете, де знаходиться кожен товар, і наведете лад на складі).

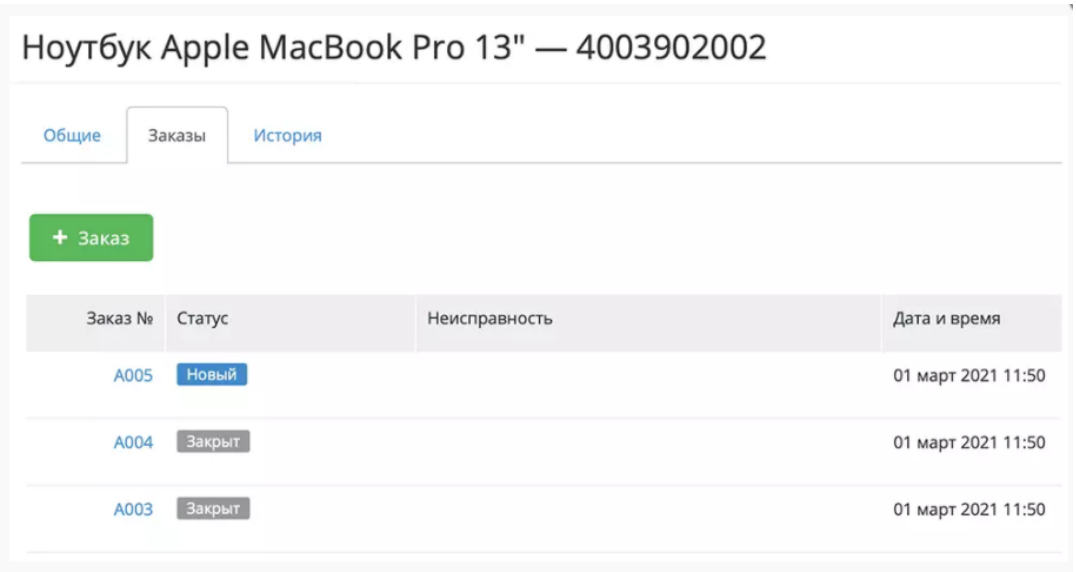

Рисунок 2.17 – Інтерфейс РемОнлайн

2.3.2 UniproRetail

Програма, яка дозволить вести повноцінний і зручний складський облік, дає власнику бізнесу та керуючому повний контроль над бізнесом. Професійна програма UniproRetail для обліку на складських об’єктах, дозволяє вести повноцінний складський облік, включаючи такі особливості:

- управління асортиментом, оптимізація складу;
- управління ціноутворенням;
- повний набір інструментів для керування роботою з постачальниками;
- звіти для аналізу всіх процесів, що відбуваються на складі.

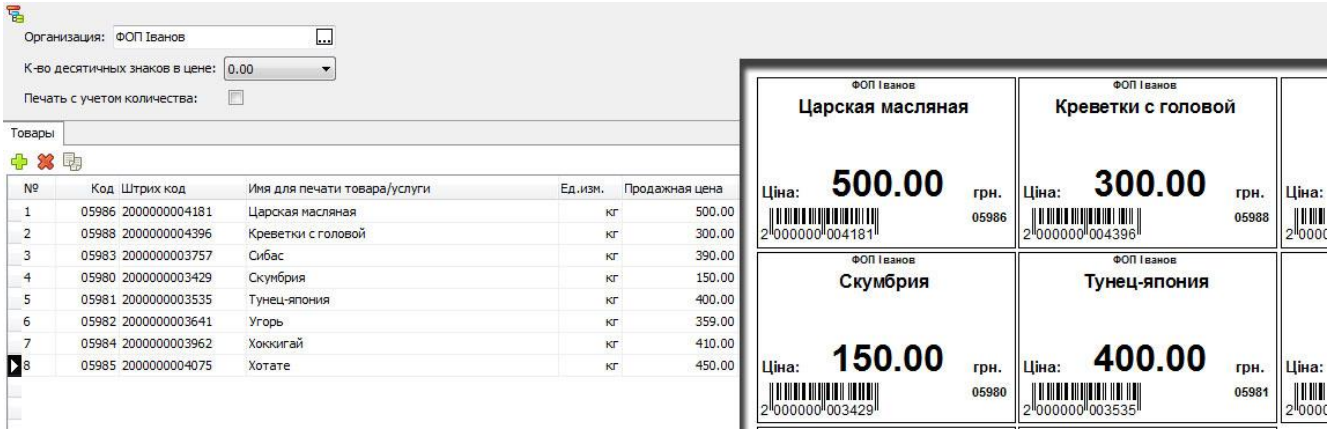

Рисунок 2.18 – Інтерфейс UniproRetail

2.3.3 jSolutions

Система jSolutions – дозволяє забезпечувати управління складською логістикою, запасами, реалізацією товарів і послуг в різних масштабах, від міні-складу до великих складських центрів. Рішення побудоване на базі хмарної платформи jSolutions і має функції, які дозволяють об'єднати в єдиній базі даних складський облік товарів, в розрізі характеристик і функцій:

- номенклатури;
- обліку партій;
- штрих-кодів товарів;
- обліку термінів придатності;

– обліку упаковок, серій, тари;

– кодів товарів згідно з УКТ ЗЕД;

– можливість обліку медикаментів і запчастин;

– облік первинних документів (вхідних/вихідних);

– ведення договорів, в т.ч. з розбивкою на етапи і додаткові положення;

– створення пов'язаних ланцюжків документів;

– оформлення вихідних податкових накладних;

– формування рахунків на оплату з можливістю відправки електронної версії рахунку на e-mail контрагента прямо з програми;

– застосування знижок;

– контроль оплат;

– облік повернення товарів;

– резервування товару;

– формування відомості товарних запасів;

– моніторинг наявності на складах;

– проведення інвентаризації;

– формування журналу товарних операцій;

– акти комплектації для виробничих завдань.

Складський облік товарів в системі має можливість інтеграції з іншими модулями системи, а при інтеграції з торгово-касовим обладнанням (ваги, сканери штрих-кодів, POS-термінали, касові апарати, фіскальні реєстратори) можливе створення системи, що дозволяє автоматизувати точки продажів, магазини, супермаркети, торговельні мережі з прогнозованим бюджетом витрат. Використовуйте різне обладнання для друку етикеток, зважування, сканування товару. Поєднуючи бізнес-процеси, в інтеграції з модулем "Бухгалтерія", можна автоматично відображати інформацію в бухгалтерському обліку про оприбуткування, реалізацію та списання товарів зі складу, розрахунки з постачальниками.

2.3.4 Висновки аналізу

Існує чотири способу ведення складського обліку:

– номенклатурний (не передбачає поділу за категоріями та групами. У кожної позиції своя картка. Такий спосіб підійде компаніям з невеликим товарообігом);

– сортовий (товари зберігаються за сортами (групам або підгрупами) та найменуваннями, без урахування партій і закупівельних цін. Тобто нові одиниці додаються до залишкових позицій та змішуються з ними незалежно від постачальника, вартості та часу надходження);

– партійний (нові партії зберігаються окремо від попередніх. Для кожної партії потрібно відкрити спеціальну картку, щоб потім вносити туди інформацію про рух товарів, які до неї входять. Після продажу всіх позицій з партії карту потрібно закрити, а потім скласти акт витрачання матеріально-виробничих запасів);

– партійно-сортовий (прийняті товари обліковуються та зберігаються партіями. При цьому всередині партій позиції поділяються за сортами. Цей спосіб часто вибирають при великому асортименті товарів).

Окрім перелічених, сьогодні існує велика кількість програмного забезпечення, що оптимізує роботу складу. По більшій мірі це ПЗ намагається підстроюватись під всі способи складського обліку, що робить їх потужним складським інструментом але має суттєві недоліки.

Вартість прямо пропорційна можливостям, із-за чого якщо підприємству потрібні лише функції контролю витратних матеріалів, воно буде переплачувати за громіздке ПЗ, половиною функцій яких не буде користуватись.

Розроблений модуль вузько спеціалізується на збереження витратних матеріалів і орієнтований виключно на сортовий облік, тому його дуже вигідно залучати до виробничих складів. Можливість імпорту і експорту даних забезпечує комунікацію з іншими відділами підприємства.

2.4 Інструкція роботи користувача

Вхідною точною модулю є – авторизація. Для авторизації в системі необхідно перейти по шляху «/authorization», заповнити дані відповідної форми і натиснути кнопку увійти. Після в ходу буде здійснено перехід на головну сторінку – «Склад».

Для переходу на іншу сторінку ж навігаційна панель в шапці модулю (Рисунок 2.19).

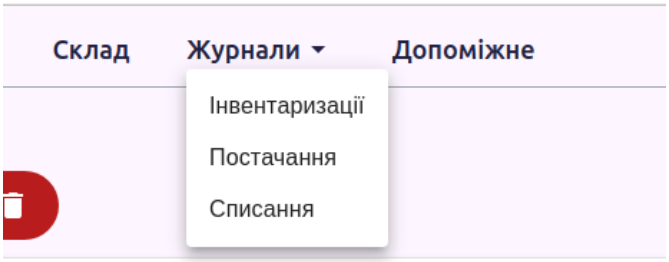

Рисунок 2.19 – Панель навігації

На сторінці «Склад» міститься перелік всіх матеріалів, і управляючий ним функціонал.

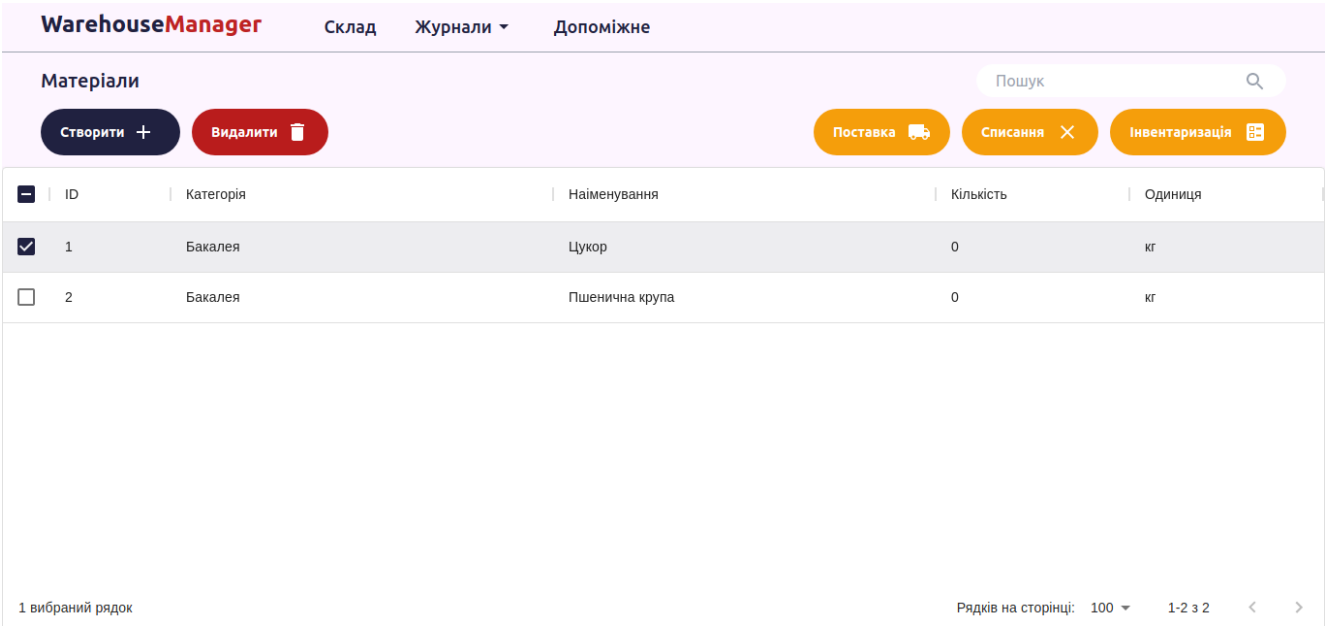

Рисунок 2.20 – Сторінка «Склад»

Меню «Журнали» містить випадаючий список, пункти якого виконують перехід до відповідного журналу.

«Допоміжне» містить в собі панелі категорій, одиниць, підстав і інтерфейс керування ними (Рисунок 2.21).

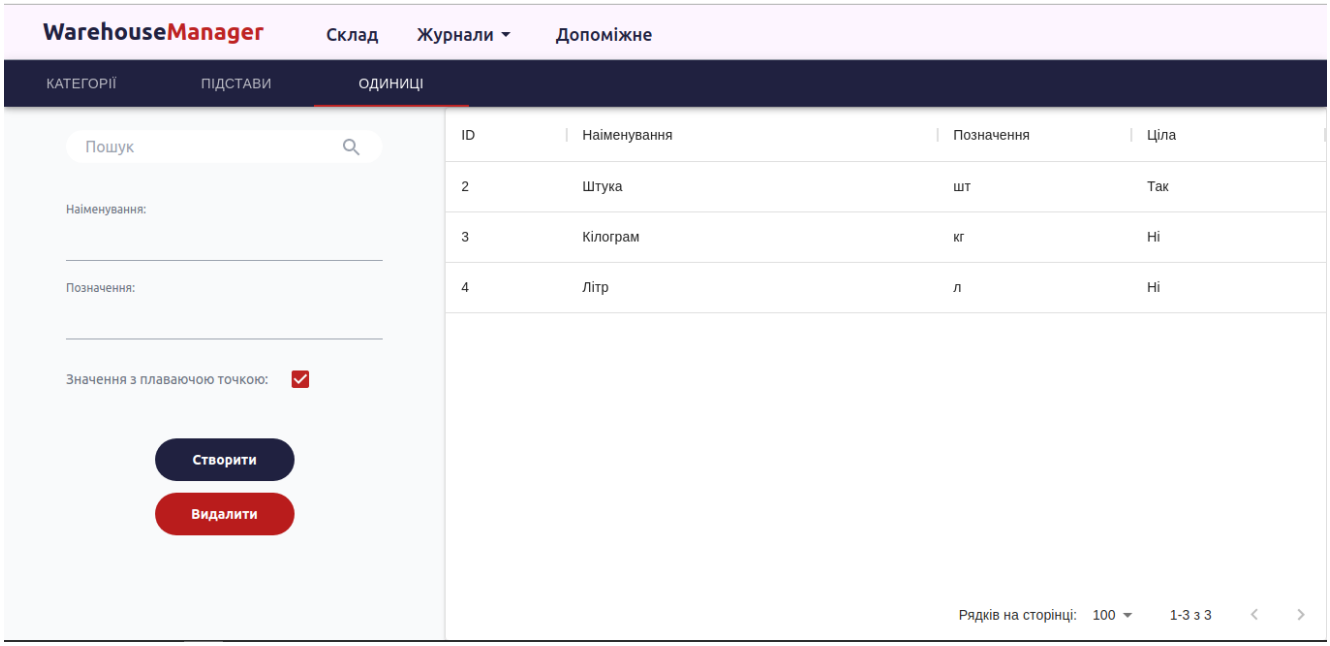

Рисунок 2.21 – Сторінка «Допоміжне»

Принцип створення нового об’єкту є легким і однаковим для всіх сутностей:

– відкрити відповідну форму (натисканням кнопки чи перейти на відповідну панель);

– заповнити всі поля даними;

– натиснути кнопку створити.

Редагування дуже подібне до створення, але передбачає вибір елементу. Для вибору елементу необхідно лише натиснути на потрібний в таблиці рядок і повторити попередні кроки. Замість введення нових даних, потрібно буде лише оновити дані в необхідних полях.

Для видалення, як і для редагування, потрібно обрати елемент і натиснути відповідну кнопку видалення. Для виконання видалення необхідно підтвердити дію, і в разі необхідності, діяти за інструкцією (для пов’язаних сутностей).

Створення задачі виконується натисканням на відповідну кнопку на сторінці «Склад». На відкритій сторінці необхідно натиснути «Додати матеріали» на нижній панелі і у відкритому модальному вікні виставити галочки в компоненті checkbox необхідних матеріалах. Виставити потрібну кількість напроти відповідних матеріалів і натиснути кнопку «Створити акт».

2.5 Тестування реалізованого програмного засобу

Таблиця 2.1 – Тест-кейси авторизації в системі

Дія	Очікуваний результат
Ввести дані існуючого облікового запису оператору	Виконана авторизація в системі і перехід до складу
Ввести дані існуючого облікового запису адміністратора	Виконана авторизація в системі і перехід до складу
Ввести дані неіснуючого облікового запису	Сповіщення про помилку

Таблиця 2.2 – Тест-кейси створення об’єкту

Дія	Очікуваний результат
5 раз ввести різні варіанти вхідних даних і натиснути кнопку «Створити»	В системі з’являться відповідні об’єкти, з’явиться повідомлення о успіху операції
5 раз зробити спробу створення об’єкту з одним пустим значенням по-черзі для кожного поля	Внизу форми з’явиться повідомлення про необхідність введення даних
5 раз зробити спробу створення об’єкту з пустими значеннями довільних полів	Внизу форми з’явиться повідомлення про необхідність введення даних
5 раз зробити спробу створити об’єкт з значенням полю найменування, яке дорівнює значенню вже існуючого об’єкта	Сповіщення про помилку

Таблиця 2.3 – Тест-кейси редагування об’єкту

Дія	Очікуваний результат
5 раз обрати об’єкт і змінивши значення його полів, натиснути кнопку редагувати	Об’єкт успішно оновиться, з’явиться повідомлення о успіху операції
5 раз обрати об’єкт і по-черзі для всіх його полів робити спробу редагування, видаливши значення	Внизу форми з’явиться повідомлення про необхідність введення даних
5 раз обрати об’єкт, змінити значення його найменування на вже існуюче і зробити спробу редагування	Сповіщення про помилку

Таблиця 2.4 – Тест-кейси видалення об’єкту

Дія	Очікуваний результат
Натиснути кнопку видалення, не обравши об’єкт	Сповіщення про необхідність вибрати об’єкт
Вибрати об’єкт і натиснути кнопку видалення	Відкриття модального вікна з підтвердженням видалення

Таблиця 2.5 – Тест-кейси задач

Дія	Очікуваний результат
5 раз обрати декілька довільних матеріалів, виставити їм довільні значення і створити задачу	Значення матеріалів на складі зміняться відповідно до операції (зменшиться – у випадку списання, збільшиться – у випадку поставки, заміниться – у випадку інвентаризації на зазначене значення), в системі з'явиться відповідний журнал
Видалити декілька довільних задач	Задача і всі її записи зникнуть з системи
Відхилити декілька довільних задач	Виконається зворотнє обчислення матеріалів, а задача і записи в системі будуть відображатись як видалені

Таблиця 2.6 – Тест-кейси імпорту та експорту

Дія	Очікуваний результат
Натиснути кнопку «Імпорт з .csv» при створені задачі та обрати файл	Система підтягне матеріали (якщо такий існує в системі) і їх значення з файлу табличного формату
Натиснути кнопку «Експорт в .csv» на сторінці інвентаризації та вказати ім'я файлу	В файлі будуть записані дані відповідної інвентаризації

Всі тест-кейси стосовно створення, редагування видалення об'єктів виконувались на кожній сутності окремо.

При тестуванні модуль показав хороші результати. Критичні помилки були відсутні, а всі малі недоліки виправлені.

3 Охорона праці

3.1 Характеристика умов праці програміста

Науково-технічний прогрес вніс серйозні зміни в умови виробничої діяльності робітників розумової праці. Їх праця стала більш інтенсивним, напруженим, які вимагають значних витрат розумової, емоційної і фізичної енергії. Це зажадало комплексного рішення проблем ергономіки, гігієни і організації праці, регламентації режимів праці та відпочинку.

В даний час комп'ютерна техніка широко застосовується у всіх областях діяльності людини. При роботі з комп'ютером людина піддається дії ряду небезпечних і шкідливих виробничих факторів: електромагнітних полів (діапазон радіочастот: ВЧ, УВЧ і СВЧ), інфрачервоного і іонізуючого випромінювань, шуму і вібрації, статичної електрики і ін.

Робота з комп'ютером характеризується значною розумовою напругою і нервово-емоційним навантаженням операторів, високою напруженістю зорової роботи і достатньо великим навантаженням на м'язи рук при роботі з клавіатурою ЕОМ. Велике значення має раціональна конструкція і розташування елементів робочого місця, що важливо для підтримки оптимальної робочої пози людини-оператора.

Програміст зобов’язаний:

– піклуватися про особисту безпеку і здоров’я, а також про безпеку і здоров’я оточуючих людей у процесі виконання будьяких робіт або під час знаходження на території підприємства;

– знати і виконувати вимоги інструкцій з охорони праці і по видах робіт на своєму робочому місці;

– виконувати роботу відповідно до вимог інструкційно-технологічної карти;

– вміти користуватися засобами індивідуального і колективного захисту;

– знати і виконувати Правила поводження з устаткуванням, інвентарем, користуватися технічним паспортом на устаткування;

– знати і виконувати обов’язки з охорони праці, передбачені колективним договором (трудовим договором), правилами внутрішнього трудового розпорядку підприємства, в тому числі:

– вчасно починати і закінчувати роботу, дотримуватися розкладу, технологічної і обідньої перерв;

– не виконувати роботи, що не передбачені змінним завданням;

– не перебувати на роботі в неробочій час без відповідного розпорядження керівника;

– дотримуватись правил корпоративного поводження;

– проходити в установленому порядку медичні огляди;

– вміти надати першу допомогу потерпілому від нещасного випадку;

– перед початком роботи перевіряти справність устаткування, огороджень, інженерно-технічних засобів безпеки, інвентарю, засобів пожежогасіння;

– співпрацювати з роботодавцем у справі організації безпечних і нешкідливих умов праці, особисто вживати можливих заходів щодо усунення будь-якої ситуації, що створює загрозу її життю чи здоров’ю або людям, які її оточують та навколишньому природному середовищу;

– при виявленні недоліків чи небезпеки зобов’язана повідомляти безпосереднього керівника або іншу посадову особу.

3.2 Вимоги до виробничих приміщень

3.2.1 Забарвлення і коефіцієнти віддзеркалення

Джерела світла, такі як світильники і вікна, які дають віддзеркалення від поверхні екрану, значно погіршують точність знаків і тягнуть за собою перешкоди фізіологічного характеру, які можуть виразитися в значній напрузі, особливо при

тривалій роботі. Віддзеркалення, включаючи віддзеркалення від вторинних джерел світла, повинне бути зведено до мінімуму. Для захисту від надмірної яскравості вікон можуть бути застосовані штори і жалюзі.

У приміщеннях, де знаходиться комп'ютер, необхідно забезпечити наступні величини коефіцієнта віддзеркалення:

- для стелі: 60-70%;
- для стін: 40-50%;
- для підлоги: приблизно 30%;
- для інших поверхонь і робочих меблів: 30-40%.

Забарвлення приміщень і меблів повинні сприяти створенню сприятливих умов для зорового сприйняття, гарного настрою. Залежно від орієнтації вікон рекомендується наступна фарбування стін і підлоги.

Якщо вікна орієнтовані на південь:

- стіни зеленувато-блакитного або світло-блакитного кольору;
- підлога зеленого кольору.

Вікна орієнтовані на північ:

- стіни світло-оранжевого або оранжево-жовтого кольору;
- підлога червонувато-оранжевого кольору.

Вікна орієнтовані на схід:

- стіни жовто-зеленого кольору;
- підлога зелена або червонувато-оранжевого кольору.

Вікна орієнтовані на захід:

- стіни жовто-зеленого або голубувато-зеленого кольору;
- підлога зелена або червонувато-оранжевого кольору.

3.2.2 Освітлення

Правильно спроектоване і виконане виробниче освітлення покращує умови зорової роботи, знижує стомлюваність, сприяє підвищенню продуктивності праці,

благотворно впливає на виробниче середовище, надаючи позитивну психологічну дію на працюючого, підвищує безпеку праці і знижує травматизм.

Недостатність освітлення приводить до напруги зору, ослабляє увагу, приводить до настання передчасної стомленості. Надмірно яскраве освітлення викликає засліплення, роздратування і різь в очах. Неправильний напрямок світла на робочому місці може створювати різкі тіні, відблиски, дезорієнтувати працюючого. Всі ці причини можуть призвести до нещасного випадку або профзахворювань, тому такий важливий правильний розрахунок освітленості.

Існує три види освітлення - природне, штучне і поєднане (природне і штучне разом).

Природне освітлення – освітлення приміщень денним світлом, що потрапляє через світлові прорізи в зовнішніх огороджуючих конструкціях приміщення. Природне освітлення характеризується тим, що змінюється в широких межах залежно від часу дня, пори року, характеру області і ряду інших чинників.

Штучне освітлення застосовується при роботі в темний час доби і вдень, коли не вдається забезпечити нормовані значення коефіцієнта природного освітлення (похмура погода, короткий світловий день). Освітлення, при якому недостатнє за нормами природне освітлення доповнюється штучним, називається змішаним освітленням.

Штучне освітлення підрозділяється на робоче, аварійне, евакуаційне, охоронне. Робоче освітлення, у свою чергу, може бути загальним або комбінованим. Загальне - освітлення, при якому світильники розміщуються у верхній зоні приміщення рівномірно, або, як розташоване устаткування. Комбіноване - освітлення, при якому до загального додається місцеве освітлення.

Згідно СНіП II-4-79 в приміщень обчислювальних центрів необхідно застосувати систему комбінованого освітлення.

При виконанні робіт категорії високої зорової точності (найменший розмір об'єкту розрізнення 0,3 ... 0,5 мм) величина коефіцієнта природного освітлення (КЕО) повинна бути не нижче 1,5%, а при зоровій роботі середньої точності (найменший розмір об'єкту розрізнення 0,5 ... 1,0 мм) КЕО повинен бути не нижче

1,0%. В якості джерел штучного освітлення звичайно використовуються люмінесцентні лампи типа ЛБ, або ДРЛ, які попарно об'єднуються в світильники, які повинні розташовуватися рівномірно над робочими поверхнями.

Вимоги до освітленості в приміщеннях, де встановлені комп'ютери, наступні: при виконанні зорових робіт високої точності загальна освітленість повинна складати 300лк, а комбінована - 750лк; аналогічні вимоги при виконанні робіт середньої точності - 200 і 300лк відповідно.

Крім того все поле зору повинне бути освітлено достатньо рівномірно - ця основна гігієнічна вимога. Іншими словами, ступінь освітлення приміщення і яскравість екрану комп'ютера повинні бути приблизно однаковими, оскільки яскраве світло в районі периферійного зору значно збільшує напруженість очей і, як наслідок, призводить до їх швидкої стомлюваності.

3.2.3 Параметри мікроклімату

Параметри мікроклімату можуть мінятися в широких межах, у той час як необхідною умовою життєдіяльності людини є підтримка постійності температури тіла завдяки терморегуляції, тобто здатності організму регулювати віддачу тепла в навколишнє середовище. Принцип нормування мікроклімату – створення оптимальних умов для теплообміну тіла людини з навколишнім середовищем.

Обчислювальна техніка є джерелом істотних тепловиділень, що може привести до підвищення температури і зниження відносної вологості в приміщенні. У приміщеннях, де встановлені комп'ютери, повинні дотримуватися певні параметри мікроклімату. У санітарних нормах СН-245-71 встановлені величини параметрів мікроклімату, що створюють комфортні умови. Ці норми встановлюються в залежності від пори року, характеру трудового процесу і характеру виробничого приміщення (Таблиця 3.1).

Об'єм приміщень, в яких розміщені працівники обчислювальних центрів, не повинен бути меншим 19,5 $м^3$ / людини з урахуванням максимального числа

одночасно працюючих в зміну. Норми подачі свіжого повітря в приміщення, де розташовані комп'ютери (Таблиця 3.2).

Таблиця 3.1 – Параметри мікроклімату, де встановлені комп'ютери

Період року	Параметр мікроклімату	Величина
Холодний	Температура повітря в приміщенні Відносна вологість Швидкість руху повітря	22-24 ° С 40-60% до 0,1 м / с
Теплий	Температура повітря в приміщенні Відносна вологість Швидкість руху повітря	23-25 ° С 40-60% 0,1-0,2 м / с

Таблиця 3.2 – Норми подачі свіжого повітря, де розташовані комп'ютери

Характеристика приміщення	Об'ємна витрата подається в приміщення свіжого повітря, м ^{3} / на одну людину в годину
Об'єм до 20м ^{3} на особу 20 ... 40м ^{3} на особу Більш 40м ^{3} на особу	Не менше 30 Не менше 20 Природна вентиляція

Для забезпечення комфортних умов використовуються як організаційні методи (раціональна організація проведення робіт залежно від пори року і доби, чергування праці і відпочинку), так і технічні засоби (вентиляція, кондиціювання повітря, опалювальна система).

3.2.4 Шум і вібрація

Шум погіршує умови праці надаючи шкідливу дію на організм людини. Працюючі в умовах тривалої шумової дії випробовують дратівливість, головні болі, запаморочення, зниження пам'яті, підвищену стомлюваність, зниження апетиту, біль у вухах тощо. Такі порушення в роботі ряду органів і систем

організму людини можуть викликати негативні зміни в емоційному стані людини аж до стресових. Під впливом шуму знижується концентрація уваги, порушуються фізіологічні функції, з'являється втома у зв'язку з підвищеними енергетичними витратами і нервово-психічним напруженням, погіршується мовна комутація. Все це знижує працездатність людини і її продуктивність, якість і безпеку праці. Тривала дія інтенсивного шуму (вище 80 дБ) на слух людини приводить до його часткової або повної втрати

Таблиця 3.3 – Граничні рівні звуку, дБ, на робочих місцях

Категорія напруженості праці	Категорія важкості праці			
	Легка	Середня	Важка	Дуже важка
Мало напружений	80	80	75	75
Помірно напружений	70	70	65	65
Напружений	60	60	-	-
Дуже напружений	50	50	-	-

Рівень шуму на робочому місці математиків-програмістів і операторів відеоматеріалів не повинен перевищувати 50дБА, а в залах обробки інформації на обчислювальних машинах - 65дБА. Для зниження рівня шуму стіни і стеля приміщень, де встановлені комп'ютери, можуть бути облицьовані звукопоглинальними матеріалами. Рівень вібрації в приміщеннях обчислювальних центрів може бути понижений шляхом встановлення устаткування на спеціальні віброізолятори.

3.2.5 Ергономічні вимоги до робочого місця

Проектування робочих місць, забезпечених відеотерміналами, відноситься до числа важливих проблем ергономічного проектування в області обчислювальної техніки.

Робоче місце і взаємне розташування всіх його елементів повинне відповідати антропометричним, фізичним і психологічним вимогам. Велике значення має також характер роботи. Зокрема, при організації робочого місця програміста повинні бути дотримані наступні основні умови: оптимальне розміщення устаткування, що до складу робочого місця і достатній робочий простір, що дозволяє здійснювати всі необхідні рухи і переміщення.

Ергономічними аспектами проектування відеотермінальних робочих місць, зокрема, є: висота робочої поверхні, розміри простору для ніг, вимоги до розташування документів на робочому місці (наявність і розміри підставки для документів, можливість різного розміщення документів, відстань від очей користувача до екрану, документа, клавіатури і т.д.), характеристики робочого крісла, вимоги до поверхні робочого столу, регульованість елементів робочого місця.

Головними елементами робочого місця програміста є стіл і крісло. Основним робочим положенням є положення сидячи.

Робоча поза сидячи викликає мінімальне стомлення програміста. Раціональне планування робочого місця передбачає чіткий порядок і сталість розміщення предметів, засобів праці і документації. Те, що потрібно для виконання робіт частіше, розташоване в зоні легкої досяжності робочого простору.

Моторне поле – простір робочого місця, в якому можуть здійснюватися рухові дії людини.

Максимальна зона досяжності рук – це частина моторного поля робочого місця, обмеженого дугами, описуваними максимально витягнутими руками при русі їх у плечовому суглобі.

Оптимальна зона – частина моторного поля робочого місця, обмеженого дугами, описуваними передпліччями при русі в ліктьових суглобах з опорою в точці ліктя і з відносно нерухомим плечем.

3.3 Розрахунок освітленості і рівня шуму

3.3.1 Розрахунок освітленості

Розрахунок освітленості робочого місця зводиться до вибору системи освітлення, визначенню необхідного числа світильників, їх типа і розміщення.

Звичайне штучне освітлення виконується за допомогою електричних джерел світла двох видів: ламп розжарювання і люмінесцентних ламп. Будемо використовувати люмінесцентні лампи, які в порівнянні з лампами розжарювання мають багато переваг:

– по спектральному складу світла вони близькі до денного, природного світла;

– володіють вищим ККД (у 1,5-2 рази вище, ніж ККД ламп розжарювання);

– володіють підвищеною світловидатністю (у 3-4 рази вище, ніж біля ламп розжарювання);

– триваліший термін служби.

3.3.2 Розрахунок рівня шуму

Одним з несприятливих чинників виробничої середи є високий рівень шуму, що створюється друкарськими пристроями, устаткуванням для кондиціювання повітря, вентиляторами систем охолоджування в самих ЕОМ.

Для вирішення питань про необхідність і доцільність зниження шуму необхідно знати рівні шуму на робочому місці оператора.

Рівень шуму, що виникає від декількох некогерентних джерел, працюючих одночасно, підраховується на підставі принципу енергетичного сумування випромінювань окремих джерел.

Звичайне робоче місце оператора оснащено наступним устаткуванням: вінчестер в системному блоці, вентилятор(и) систем охолоджування ПК, монітор, клавіатура, принтер і сканер.

Таблиця 3.4 – Джерела шуму

Джерело шуму	Рівень шуму, дБ
Жорсткий диск	15
Кулер	29
Монітор	0
Принтер	63
Сканер	38
Кондиціонер	36
Клавіатура	10

Підставивши значення рівня звукового тиску для кожного виду обладнання у формулу, отримаємо 49,5 дБ.

Отримане значення не перевищує допустимий рівень шуму для робочого місця оператора, рівний 65 дБ (ГОСТ 12.1.003-83). І якщо врахувати, що навряд чи такі периферійні пристрої як сканер і принтер будуть використовуватися одночасно, то ця цифра буде ще нижчою. Крім того при роботі принтера безпосередню присутність оператора необов’язково, тому що принтер обладнаний механізмом автоподачі аркушів.

3.4 Заходи та засоби протипожежного захисту

Відповідно до нормативних документів пожежна безпека об'єкта повинна забезпечуватися системою запобігання пожежі, системою протипожежного захисту і системою організаційно-технічних заходів.

Організаційні заходи пожежної безпеки передбачають: організацію пожежної охорони на об’єкті, проведення навчань з питань пожежної безпеки (включаючи інструктажі та пожежно-технічні мінімуми), застосування наочних засобів протипожежної пропаганди та агітації, організацію добровільних пожежних дружин та пожежно-технічної комісії, проведення перевірок, оглядів стану пожежної безпеки приміщень, будівель, об'єкта в цілому тощо.

До технічних заходів належать: суворе дотримання правил і норм, визначених чинними нормативними документами при реконструкції приміщень, будівель та об’єктів, технічному переоснащенні виробництва, експлуатації чи можливому переобладнанні електромереж, опалення, вентиляції, освітлення тощо.

Заходи режимного характеру передбачають заборону куріння та застосування відкритого вогню в недозволених місцях, недопущення появи сторонніх осіб у вибухонебезпечних приміщеннях чи об’єктах, регламентацію пожежної безпеки при проведенні вогневих робіт тощо.

Експлуатаційні заходи включають своєчасне проведення профілактичних оглядів, випробувань, ремонтів технологічного та допоміжного устаткування, а також інженерного господарства (електромереж, електроустановок, опалення, вентиляції).

Комплекс заходів, спрямованих на ліквідацію пожежі, що виникла, є пожежогасінням. Основою пожежогасіння є примусове припинення процесу горіння одним з таких способів:

спосіб охолодження ґрунтується на тому, що горіння речовини можливе тільки тоді, коли температура її верхнього шару вища за температуру його запалювання. Якщо з поверхні горючої речовини відвести тепло, тобто охолодити її нижче температури запалювання, горіння припиняється;

– спосіб розведення базується на здатності речовини горіти при вмісті кисню у атмосфері більше 14-16% за об'ємом. Зі зменшенням кисню в повітрі нижче вказаної величини полум'яне горіння припиняється, а потім припиняється і тління внаслідок зменшення швидкості окислення. Зменшення концентрації кисню досягається введенням у повітря інертних газів та пари із зовні або розведенням кисню продуктами горіння (у ізольованих приміщеннях);

– спосіб ізоляції ґрунтується на припиненні надходження кисню повітря до речовини, що горить. Для цього застосовують різні ізолюючі вогнегасні речовини (хімічна піна, порошок та інше);

– спосіб хімічного гальмування реакцій горіння полягає у введенні в зону горіння галоїдно-похідних речовин (бромисті метил та етил, фреон та інше), які при

потраплянні у полум'я розпадаються і з'єднуються з активними центрами, припиняючи екзотермічну реакцію, тобто виділення тепла. У результаті цього процес горіння припиняється;

– спосіб механічного гасіння полум'я сильним струменем води, порошку чи газу;

– спосіб вогнеперешкоди заснований на створенні умов, за яких полум'я не поширюється через вузькі канали, переріз яких менше критичного.

Реалізація способів припинення горіння досягається використанням вогнегасних речовин та технічних засобів. До вогнегасних належать речовини, що мають фізико-хімічні властивості, які дозволяють створювати умови для припинення горіння. Серед них найпоширенішими є вода, водяна пара, піна, газові вогнегасні склади, порошки, пісок, пожежостійкі тканини тощо.

Кожному способу припинення горіння відповідає конкретний вид вогнегасних засобів. Наприклад, для охолоджування використовують воду, водні розчини, снігоподібну вуглекислоту; для розведення горючого середовища – діоксид вуглецю, інертні гази, водяну пару; для ізоляції вогнища – піну, пісок; хімічне гальмування горіння здійснюється за допомогою брометилу, хладону, спеціальних порошків.

Вибір вогнегасної речовини залежить від характеру пожежі, властивостей і агрегатного стану речовин, що горять, параметрів пожежі (площі, інтенсивності, температури горіння тощо), виду пожежі (у закритому або відкритому повітрі), вогнегасної здатності щодо гасіння конкретних речовин та матеріалів, ефективності способу гасіння пожежі.

Оскільки вода є основною вогнегасною речовиною, необхідно приділити особливу увагу створенню та працездатності надійних систем водопостачання.

Відповідно до протипожежних норм, кожне промислове підприємство обладнують пожежним водопроводом. Він може бути об'єднаним з господарсько-питним або водопроводом, який використовують у виробничому процесі. Воду також можна подавати до місця пожежі з водоймищ річок або підвозити в автоцистернах.

Будинки, споруди, приміщення, технологічні установки повинні бути забезпечені первинними засобами пожежогасіння: вогнегасниками, ящиками з піском, покривалами з негорючого теплоізоляційного полотна, грубововняної тканини чи повсті, іншим пожежним інструментом, які використовуються для локалізації і ліквідації пожеж у початковій стадії їхнього розвитку.

Норми належності первинних засобів пожежогасіння для об'єктів необхідно встановлювати згідно з нормами технологічного проектування, Типовими нормами належності вогнегасників (НАПБ Б.ОЗ.001-2004) та Правилами пожежної безпеки в Україні.

Коли від пожежі захищаються приміщення з персональними комп’ютерами, то необхідно враховувати специфіку вогнегасних речовин у вогнегасниках, які призводять під час гасіння до псування обладнання. Ці приміщення рекомендується оснащувати вуглекислотними вогнегасниками з урахуванням граничнодопустимої концентрації вогнегасної речовини.

Для зазначення місцезнаходження первинних засобів пожежогасіння необхідно встановлювати відповідні знаки згідно з чинними державними стандартами. Знаки необхідно розміщувати на видних місцях на висоті від 2 м до 2,5 м від рівня підлоги як усередині, так і поза приміщеннями (у разі потреби).

Переносні вогнегасники повинні розміщуватися шляхом:

– навішування на вертикальні конструкції на висоті не більше 1,5 м від рівня підлоги до нижнього торця вогнегасника і на відстані від дверей, достатній для її повного відчинення;

– установлення в пожежні шафи пожежних кранів, або у спеціальні тумби;

3.5 Висновки до розділу

В даному розділі були розглянуті питання стосовно безпеки праці програміста. Дотримання зазначених в цьому розділі правил та вимог, забезпечить комфортний, а головне безпечний, робочий процес.

Висновки

В ході виконання дипломного проекту було створено модуль, що оптимізує процес складського обліку, підвищуючи ефективність роботи його працівників.

Розроблений модуль реалізує сортовий вид обліку, тому орієнтований в більшій мірі на склади, що зберігають витратні матеріали. Можливість імпорту і експорту даних, дозволяє модулю взаємодіяти з іншими системами чи модулями підприємства.

При тестуванні, програмний засіб дав хороші результати. Всі задачі складського обліку він виконує без помилок.

Передбачення можливих шляхів шахрайства в межах складу допомогли спроектувати надійну систему журналів, що відображає всі дії працівників на складі. А авторизація в системі запобігає впливу і доступу до інформації невповноважених осіб. Запобігання помилковим діям користувача гарантує безпеку від тяжких наслідків.

Звісно, модуль ще не є досконалим, і він потребує вдосконалення (наприклад на даний момент він має не гнучку систему одиниць вимірювання), але при продовженні підтримки і модернізації, він стане потужним конкурентоспроможним гравцем на ринку.

Плани на майбутнє:

- розробка більш гнучкої системи одиниць вимірювання;
- введення системи рецептів (внесення в систему переліку матеріалів, що витрачаються на одиницю товару) дозволить при закінченні робочого дня розрахувати витрачені матеріали, що підвищить ефективність використання програмного засобу на виробничих складах;
- введення системи управління користувачами, що дозволить прямо в програмному засобі керівнику здійснювати управління обліковими записами і імпортувати їх з головної системи підприємства.

Перелік використаних джерел

1. Буч Г. Язык UML. Руководство пользователя / Грейди Буч, Джеймс Рамбо, Айвар Джекобсон: Пер. с англ. Слинкин А. А. – 2-е изд., стер. – М.: ДМК Пресс; СПб.: Питер, 2004. – 432 с.: ил. – (Серия «Объектно-ориентированные технологи в програмировании»).

2. Вайнман Л, Вайнман В. Креативный Web-дизайн на HTML 4. Издательство «ДиаСофт», 2003 - 528 с;

3. Гандзюк М.П., Желібо Є.П., Халімовський М.О. Основи Охорони праці: Підруч. для студ. вищих навч. закл. За ред. М.П. Гандзюка. - К.: Каравела; Львів: Новий Світ-2000, 2003. - 408 с;

4. Глушаков С. Программирование Web-страниц. Харьков: Фолио, 2005 – 390 с;

5. Гниденко, И. Г. Технология разработки программного обеспечения : учеб. пособие для СПО / И. Г. Гниденко, Ф. Ф. Павлов, Д. Ю. Федоров. – М. : Издательство Юрайт, 2017. – 235 с;

6. Гордеев, С. И. Организация баз данных в 2 ч. Часть 2 : учебник для вузов / С. И. Гордеев, В. Н. Волошина. – 2-е изд., испр. и доп. – М. : Издательство Юрайт, 2019. – 501 с;

7. Желібо Є П., Заверуха Н. М., Зацарний В, В. Безпека життедіяльності / За ред. Є П. Желібо. - К.: Каравела, 2010. - 328 с;

8. Кубенский, А. А. Функциональное программирование : учебник и практикум для академического бакалавриата / А. А. Кубенский. – М. : Издательство Юрайт, 2019. – 348 с;

9. Кудрявцев, К.Я. Методы оптимизации: учеб. пособие для вузов / К.Я. Кудрявцев, А.М. Прудников. – 2-е изд. – М.: Издательство Юрайт, 2019. – 140 с;

10. Ламан К. Применение UML и шаблонов проєктирования. Издательский дом «Вильямс», 2004 – 624 с;

11. Мак-Фердис. Использование JavaScript. Специальное издание. Издательский дом «Вильямс», 2002 – 896 с;

12. Ратишллер Т., Геркен Т. PHP4. Разработка Web-приложений. Питер, 2001 – 384 с;

13. Роберт У. Себеста. Основные концепции языков программирования. 5-е издание. 668 с;

14. Стасышин, В. М. Базы данных: технологии доступа : учеб. пособие для СПО / В. М. Стасышин, Т. Л. Стасышина. — 2-е изд., испр. и доп. — М. : Издательство Юрайт, 2018. — 164 с;

15. Трофимов, В. В. Основы алгоритмизации и программирования : учебник для СПО / В. В. Трофимов, Т. А. Павловская ; под ред. В. В. Трофимова. — М. : Издательство Юрайт, 2019. — 137 с;

16. Черткова, Е. А. Статистика. Автоматизация обработки информации : учеб. пособие для вузов / Е. А. Черткова ; под общ. ред. Е. А. Чертковой. — 2-е изд., испр. и доп. — М. : Издательство Юрайт, 2017. — 195 с;

Додаток А – Текст програми

```
import axios from "../axios";
export const getMaterials = async (query = "") =>
  (await axios.get(`materials${query}`)).data;
export const getMaterial = async (id) =>
  (await axios.get(`materials/${id}`)).data;
export const getMaterialsCount = async () =>
  (await axios.get("materials/count")).data;
export const postMaterial = async (material) =>
  (await axios.post("materials", { ...material })).data;
export const putMaterial = async ({ id, ...material }) =>
  (await axios.put(`materials/${id}`, material)).data;
export const deleteMaterial = async (id) => {
  return (await axios.delete(`materials/${id}`)).data;};
import { useMaterialsContext } from "@/src/providers/materials";
import { useCategoriesContext } from "@/src/providers/categories";
import { useScalesContext } from "@/src/providers/scales";
import { usePageContext } from "@/blocks/context";
import { useNotificaitonsContext } from "@/src/providers/notifications";
import { useState, useEffect } from "react";
import { useForm } from "react-hook-form";
import { find, propEq } from "rambda";
import Submit from "./submit-button";
import TextInput from "@/components/TextInput";
import TextField from "@material-ui/core/TextField";
import { propEq, reject } from "ramda";
import { useCallback, useState } from "react";
import {
  deleteMaterial,
  getMaterials,
  postMaterial,
  putMaterial,
```

```
} from "../api/material";
import { alter } from "../helpers/array";
const useMaterials = () => {
  const [materials, setMaterials] = useState([]);
  const [selected, setSelected] = useState([1, 2]);
  return {
    materials,
    setMaterials,
    selected,
    setSelected,
    retrieve: useCallback(
      async (query = "") => setMaterials(await getMaterials(query)),
      []),
    remove: useCallback(
      async (id) => {
        await deleteMaterial(id);
        setMaterials([...reject(propEq("id", id), materials)]);
      },[materials]),
    create: useCallback(
      async (data) => {
        const material = await postMaterial(data);
        setMaterials([...materials, { ...material }]);
        return material; }, [materials]),
    update: useCallback(
      async (payload) => {
        const material = await putMaterial(payload);
        setMaterials(
          alter(propEq("id", material.id), materials, (m) => ({
            ...m,
            ...material,
          })));
        return material; }, [materials]),};};
import { createContext, useContext } from "react";
import useMaterials from "../../hooks/useMaterials";
```

```
export const MaterialsContext = createContext({});
export const useMaterialsContext = () => useContext(MaterialsContext);

const MaterialsProvider = ({ children }) => (
  <MaterialsContext.Provider value={useMaterials()}>
    {children}
  </MaterialsContext.Provider>
);
export default MaterialsProvider;
export default useMaterials;
export default function CreateOrUpdateMaterial({
  target = null,
  setTarget = () => null,
}) {
  const {
    register,
    handleSubmit,
    setValue,
    watch,
    formState: { errors },
  } = useForm();
  const { materials, create, update } = useMaterialsContext();
  const { categories } = useCategoriesContext();
  const { scales } = useScalesContext();
  const { success, error, warning } = useNotificaitonsContext();
  const [material, setMaterial] = useState(null);
  useEffect(() => {
    setMaterial(find(propEq("id", target), materials));
  }, [target]);
  useEffect(() => {
    var defaultCategory = null,
      defaultScale = null;
    if (categories.length > 0) defaultCategory = categories[0].id;
    if (scales.length > 0) defaultScale = scales[0].id;
    setValue("id", material ? material.id : null);
```

```
  setValue("name", material ? material.name : null);
  setValue(
    "category",
    material
      ? material.category
        ? material.category.id
        : defaultCategory
      : defaultCategory
  );
  setValue("value", material ? material.value : 0);
  setValue(
    "scale",
    material
      ? material.scale
        ? material.scale.id
        : defaultScale
      : defaultScale
  );
}, [material]);

const onSubmit = (data, e) => {
  if (material) {
    update(data)
      .then(() => {
        success(() => <p>Матеріал успішно оновлено</p>);
        setTarget(null);
      })
      .catch((e) => {
        error(() => (
          <>
            <p className="font-bold">
              Сталася помилка під час оновлення матеріалу
            </p>
            <p>Код помилки: {e.response && e.response.status}</p>
          </>
```

```
        ));
      });
    } else {
      create(data)
        .then(() => {
          success(() => <p>Матеріал успішно створено</p>);
          setTarget(null);
        })
        .catch((e) =>
          error(() => (
            <>
              <p className="font-bold">
                Сталася помилка під час створення нового матеріалу
              </p>
              <p>Код помилки: {e.response && e.response.status}</p>
            </>
          ))
        );
    }
    e.target.reset && e.target.reset();
  };

  return (
    <div>
      <form
        onSubmit={handleSubmit(onSubmit)}
        className="flex flex-col w-full space-y-5"
      >
        <div className="flex flex-row">
          <TextInput
            label="Наіменування:"
            type="text"
            defaultValue={material ? material.name : ""}
            {...register("name", { required: true })}
            onChange={(e) => {
```

```
      setValue("name", e.value);
    }}
  />
</div>
<div className="flex flex-row">
  <TextField
    className="w-full"
    select
    label="Категорія"
    value={watch("category")}
    {...register("category", { required: true })}
    onChange={(e) => {
      setValue("category", e.target.value);
    }}
    SelectProps={{
      native: true,
    }}
  >
    {categories.map((option) => (
      <option key={option.id} value={option.id}>
        {option.name}
      </option>
    ))}
  </TextField>
</div>
<div className="flex flex-row">
  <TextField
    className="w-full"
    select
    label="Одиниця"
    value={watch("scale")}
    {...register("scale", { required: true })}
    onChange={(e) => {
      setValue("scale", e.target.value);
    }}
```

```
          SelectProps={{
            native: true,}}>
          {scales.map((option) => (
            <option key={option.id} value={option.id}>
              {option.symbol}
            </option>
          ))}
        </TextField>
      </div>
      <div className="flex flex-row justify-center">
        <Submit object={target} />
      </div>
      <div className="flex flex-row text-xs text-red-700">
        {errors.name && <span>введіть наіменування...</span>}
      </div>
    </form>
  </div>);}
import { usePageContext } from "@/blocks/context";
import { useState } from "react";
import ModalWindow from "@/components/ModalWindow";
import Button from "@/components/Button";
import AddIcon from "@material-ui/icons/Add";
import EditIcon from "@material-ui/icons/Edit";
import CreateOrUpdateMaterial from "@/components/forms/create-or-update/material";
export default function Create() {
  const { target, setTarget } = usePageContext();
  const [open, setOpen] = useState(false);
  const onClose = () => {
    setOpen(false);
  };
  return (
    <div>
      <Button
        onClick={(e) => {
          setOpen(true);
```

```
        }}
        color={
          target
            ? "w-36 bg-green-800 hover:bg-green-700"
            : "w-36 bg-primary-main hover:bg-primary-light"}>
        <div className="flex flex-row items-center space-x-2">
          <span>{target ? "Редагувати" : "Створити"}</span>
          {target ? <EditIcon /> : <AddIcon />}
        </div>
      </Button>
      <ModalWindow open={open} onClose={onClose}>
        <CreateOrUpdateMaterial target={target} setTarget={setTarget} />
      </ModalWindow>
    </div>
  );
}
import { useMaterialsContext } from "@/src/providers/materials";
import { useCategoriesContext } from "@/src/providers/categories";
import { useScalesContext } from "@/src/providers/scales";
import { usePageContext } from "@/blocks/context";
import { useNotificaitonsContext } from "@/src/providers/notifications";
import { useEffect } from "react";
import { find, propEq } from "rambda";
import DataView from "@/components/DataView";
import { view } from "rambda";

const columns = [
  { field: "id", headerName: "ID", width: 125 },
  { field: "category", headerName: "Категорія", flex: true },
  { field: "name", headerName: "Наіменування", flex: true },
  { field: "value", headerName: "Кількість", width: 200 },
  { field: "scale", headerName: "Одиниця", width: 200 },
];
export default function DataTable() {
  const {
```

```
  materials,
  retrieve: getMaterials,
  setSelected,
  selected,
} = useMaterialsContext();
const { retrieve: getCategories } = useCategoriesContext();
const { retrieve: getScales } = useScalesContext();
const { error } = useNotificaitonsContext();
const { view, setView, target, setTarget } = usePageContext();
function TransferToView(array) {
  var result = [];
  array.forEach((element) => {
    console.log(element);
    result.push({
      id: element.id,
      name: element.name,
      category: element.category ? element.category.name : null,
      scale: element.scale ? element.scale.symbol : null,
      value: element.value,
    });
  }); return result;}
useEffect(() => {
  getMaterials().catch((e) => {
    error(() => (
      <>
        <p>При спробі зчитати матеріали виникла помилка</p>
        <p>Код помилки: {e.response && e.response.status}</p>
      </>
    ));
  });
  getCategories().catch((e) => {
    error(() => (
      <>
        <p>При спробі зчитати категорії виникла помилка</p>
        <p>Код помилки: {e.response && e.response.status}</p>
```

```
      </>
    ));
   });
   getScales().catch((e) => {
    error(() => (
     <>
      <p>При спробі зчитати одиниці виникла помилка</p>
      <p>Код помилки: {e.response && e.response.status}</p>
     </>
    ));
   });
  }, []);
  useEffect(() => {
   setView([...materials]);
  }, [materials]);
  return (
   <DataView
    columns={columns}
    rows={view}
    setTarget={setTarget}
    setSelected={setSelected}
    target={target}
    TransferToView={TransferToView}
    checkboxSelection
   />);}
import { useMaterialsContext } from "@/src/providers/materials";
import { useNotificaitonsContext } from "@/src/providers/notifications";
import { usePageContext } from "@/blocks/context";
import { useState } from "react";
import Button from "@/components/Button";
import ModalWindow from "@/components/ModalWindow";
import DeleteIcon from "@material-ui/icons/Delete";
export default function Delete() {
  const { remove } = useMaterialsContext();
  const { target, setTarget } = usePageContext();
```

```
const { info, success, error } = useNotificaitonsContext();
const [open, setOpen] = useState(false);
const onClose = () => {
  setOpen(false);
};
const onDelete = () => {};
return (
  <div className="flex flex-row">
    <Button
      onClick={() => {
        target
          ? setOpen(true)
          : info(() => <span>Оберіть матеріал для видалення</span>);
      }}
      color="bg-red-700 hover:bg-red-600"
    >
      <div className="flex flex-row items-center space-x-2">
        <span>Видалити</span>
        <DeleteIcon />
      </div>
    </Button>
    <ModalWindow open={open} onClose={onClose}>
      <div className="flex flex-col space-y-5">
        <div className="flex flex-row justify-center">
          <span>Ви впевнені?</span>
        </div>
        <div className="flex flex-row justify-center space-x-5">
          <Button
            onClick={onDelete}
            color="w-24 bg-primary-main hover:bg-primary-light"
          >
            Так
          </Button>
          <Button onClick={onClose} color="w-24 bg-red-700 hover:bg-red-600">
            Ні
```

```
          </Button>
        </div>
      </div>
    </ModalWindow>
  </div>
 );
}
import { useMaterialsContext } from "@/src/providers/materials";
import PageContext from "../context";
import { useState } from "react";
import Header from "../../components/Header";
import Search from "@/components/Search";
import DataTable from "./parts/data-table";
import Create from "./parts/create";
import Delete from "./parts/delete";
import CreateSupply from "./parts/createSupply";
import CreateInventory from "./parts/createInventory";
import CreateCancellation from "./parts/createCancellation";

export default function MaterialsBlock() {
 const { materials } = useMaterialsContext();
 const [selected, setSelected] = useState([]);
 const [target, setTarget] = useState(null);
 const [view, setView] = useState([]);

 return (
  <PageContext.Provider
   value={{ view, setView, target, setTarget, selected, setSelected }}
  >
   <div className="flex flex-col w-screen h-screen max-w-screen max-h-screen">
    <Header />
    <div className="flex flex-col px-10 py-3 space-y-3">
     <div className="flex flex-row items-center justify-between">
      <span className="text-primary-main text-xl font-medium">
       Матеріали
```

```
            {target ? " / " + target : ""}
          </span>
          <Search array={materials} setView={setView} />
        </div>
        <div className="flex flex-row items-center justify-between">
          <div className="flex flex-row space-x-3 items-center">
            <Create />
            <Delete />
          </div>
          <div className="flex flex-row space-x-3 items-center">
            <CreateSupply />
            <CreateCancellation />
            <CreateInventory />
          </div>
        </div>
      </div>
      <div className="flex w-full h-full">
        <DataTable />
      </div>
    </div>
  </PageContext.Provider>
 );
}
```