ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської кваліфікаційної роботи

на ступінь вищої освіти бакалавр

на тему: «Розробка web-додатку для керування торгівлею та складського обліку на платформі ASP.NET Core Blazor»

Виконав: студент 4 курсу, групи ПД– 42

спеціальності

121___Інженерія програмного забезпечення

(шифр і назва спеціальності)

_____Баглай В.О._____

(прізвище та ініціали)

Керівник __Негоденко О.В._____

(прізвище та ініціали)

Рецензент_____

(прізвище та ініціали)

Нормоконтроль _____

(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти - «Бакалавр»

Спеціальність - 121 Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ

Завідувач кафедри

Інженерії програмного

забезпечення

_____О.В. Негоденко

«____»_____2022 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТУ

Баглай Владислав Олегович
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка web-додатку для керування торгівлею та складського обліку на платформі ASP.NET Core Blazor»
   Керівник роботи к.т.н., доцент Негоденко О.В.
   (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
   затверджені наказом вищого навчального закладу від "16" лютого 2022 року №22.
2. Строк подання студентом роботи 03.06.2022
3. Вихідні дані до роботи:

3.1. Офіційна документація Microsoft.
3.2. Visual Studio.
3.3. Існуючі інструменти для керування торгівлею та складським обліком
3.4. Наукова-технічна література

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

4.1. Аналіз та огляд існуючих додатків та інструментів для реалізації системи.
4.2. Розробка структури web-додатку для керування торгівлею та складським обліком.

4.3. Програмна реалізація додатку.
4.4. Висновки.
5. Перелік графічного матеріалу
5.1 Титульний слайд
5.2 Мета, об’єкт, предмет та наукова новизна дослідження
5.3 Аналоги
5.4 Технічні завдання
5.5 Програмні засоби реалізації
5.6 Діаграма прецендентів системи
5.7 Діаграма діяльності
5.8 Схема бази даних додатку
5.9 Діаграма класів
5.10 Апробація результатів дослідження
5.11 Висновки

6. Дата видачі завдання 11.03.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Вибір теми бакалаврської роботи	11.03.2022	Виконано
2	Підбір науково-технічної літератури	12.03.2022-20.03.2022	Виконано
3	Дослідження аналогів та актуальності додатку	20.03.2022-04.04.2022	Виконано
4	Аналіз та вибір інструментів для розробки додатку	04.04.2022-25.04.2022	Виконано
5	Проектування та реалізація	25.04.2022-06.05.2022	Виконано
6	Висновки, оформлення роботи	06.05.2022-13.05.2022	Виконано
7	Передзахист	16.05.2022-01.06.2022	
8	Захист роботи		

Студент _____ В.О. Баглай
(підпис) (прізвище та ініціали)

Керівник роботи_____ О.В. Негоденко

РЕФЕРАТ

Текстова частина бакалаврської роботи с., рис., джерел

Об'єкт дослідження – процес керування торгівлею та складським обліком.

Предмет дослідження – технології розробки web-додатків за допомогою платформи ASP.NET Core Blazor.

Мета дослідження – підвищити ефективність складського обліку поставки та реалізації товарів.

У роботі проведено аналіз існуючих додатків, таких як «ERP FOSS», «PERFECTUM» та «Дебет Плюс»

Загальною проблемою цих продуктів є велика кількість функціоналу яка створює лишні кроки для керування бізнес-процесами і потребує більших витрат часу.

В якості сервера БД було взято SQL Server та Entity Framework Core для роботи з нею.

Виконано опис технологій, які були використані у ході роботи, та розробки. Додаток реалізований за допомогою платформи ASP.NET Core Blazor.

Здійснено аналіз сучасних web-додатків та технологій, які використовуються для побудови.

На основі результатів виконаних досліджень розроблено концепцію web-додатка.

Упровадження розробленої схеми (методики) дозволяє пришвидшити процес обліку поставки та реалізації товарів.

Галузь використання – невеликим фірмам, приватним підприємцям у сфері роздрібної торгівлі.

ЗМІСТ

ВСТУП

Актуальність теми. На сьогоднішній день багато людей вирішують створити свій власний бізнес. З початком створення власного бізнесу підприємці-початківці стикаються з проблемами контролю основним бізнес-процесів. Тому з'являється попит на такі системи для контролю бізнес-процесів. Так як автоматизація бізнесу існує і її потрібно використовувати. Так додатки допомагають уникнути такі проблеми підприємств, як затримка, простої, недостачі, що суттєво впливає на підприємство. А також дозволяє зекономити значну частину часу для робітників на рутинних операціях і сконцентруватися на більш важливих задачах.

Об'єкт дослідження – процес керування торгівлею та складським обліком.

Предмет дослідження – технології розробки web-додатків за допомогою платформи ASP.NET Core Blazor.

Мета дослідження – підвищити ефективність складського обліку поставки та реалізації товарів.

Для виконання поставленої мети слід виконати наступні завдання:

1. Проаналізувати переваги та недоліки існуючих додатків
2. Розробити вимоги до web-додатку на основі аналізу переваг та недоліків існуючих додатків.
3. Проаналізувати технічні засоби, що використовуються для розробки та обрано необхідні для створення надійного додатку.
4. Спроектувати та розробити новий додаток на основі аналізу потреб користувачів.

Практичне значення одержаних результатів полягає у тому, що матеріли досліджень створюють підстави для подальшого вивчення використаних технологій та програмної реалізації бізнес-процесів. Теоретичні та практичні розробки досліджень можуть бути використані у сфері роздрібної торгівлі для підвищення ефективності складського обліку поставки та реалізації товарів.

Результати роботи. Матеріали дипломного проекту можуть сприяти вдосконаленню.

1.АНАЛІЗ ТА ОГЛЯД ІСНУЮЧИХ ДОДАТКІВ ТА ІНСТРУМЕНТІВ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ

1.1. Поширення бізнесу та прогноз зростання малого та середнього бізнеса

Більшість тенденцій говорять про зростання та поширення малого бізнесу. Так як підприємства виступають важливим чинником стабільної державної економіки, а їх кількість та ступінь спроможності визначає, наскільки економіка країни досягла рівня соціально орієнтованого ринку.

З огляду на інформацію за 2019 рік, кількість ФОП склала 1 млн 866 000, що на 7,5% більше, ніж минулого року. Зокрема, значно зросла кількість фізичних осіб на спрощеній системі оподаткування. Загалом у порівнянні з минулим роком кількість спрощенців збільшилася майже на 93 800 осіб. Так, якщо на 1 січня 2019 року на спрощеній системі оподаткування було 1,39 мільйона осіб, то з 1 липня 2019 року їх кількість скоротилася до 1,36 мільйона. Проте станом на 1 січня поточного року кількість підприємців зросла до 1,5 мільйона. У третю групу відібрано 596 тис. осіб, що на 79 тис. більше з 1 січня минулого року.

Враховуючи ситуацію в світі із за COVID, який почався в 2020 році це був дуже сильній удар по малому бізнесу із-за обмежень, які вводили для безпеки. Велика кількість малого бізнесу призупинили роботу, або повністю закрились.

Якщо аналізувати український ринок, то станом на 13 жовтня 2021 року в Україні налічується 1 млн 982 тис. ФОП. Це майже вдвічі більше, ніж наприкінці 2020 року. Тому статистика підтверджує, що ситуація з COVID стала рушійною силою для багатьох українців створити власних приватних підприємців, стартапів.

За офіційною статистикою по ФОПам за перше півріччя 2021 року роздрібна торгівля показує відносний ріст незважаючи на те що, ситуація з COVID була складною для підприємців (в особливості роздрібної торгівлі). Також ми повинні розуміти, що це лише офіційні данні, так як значна частка підприємців не зареєстровані та не сплачують податки.

У 2021 році, зареєстровано 647 тис. нових підприємців розпочали свою діяльність (роздрібна торгівля), що створило 10 тис. малих бізнесів.

IT	За перше півріччя 2021 року в Україні з'явилося майже 16 тисяч нових ФОП працівників IT галузі (розробників, програмістів, HR, рекрутерів, дизайнерів, маркетологів, і т.д.). Їх кількість зросла на 9% й зараз становить 188 966 ФОП загалом.
Роздрібна торгівля	У цій сфері за перше півріччя 2021 року в Україні було зареєстровано 647 716 нових ФОП — це на 10 155 бізнесів більше, ніж на початку 2021 року. Відносний ріст за цей час становить 1,6%.
Надання інформаційних та індивідуальних послуг	Кількість нових ФОП, що займаються інформаційними послугами, у 2021 році в Україні зросла на 3284 — з 39 080 до 42 364 (+8%).
Забезпечення стравами та напоями	У січні 2021 році було 58 590 ФОПів у цій сфері, а наразі кількість ФОП з таким КВЕД зросла до 62 248. Відповідно, їх кількість зросла на 6% (або на 3658 компаній).

Рисунок 1.1 – Кількість ФОПів на перше півріччя 2021 року

Роздрібна торгівля займає близько 27% від усіх ФОПів, близько 27 %. Що являється значною часткою серед усіх ФОПів.

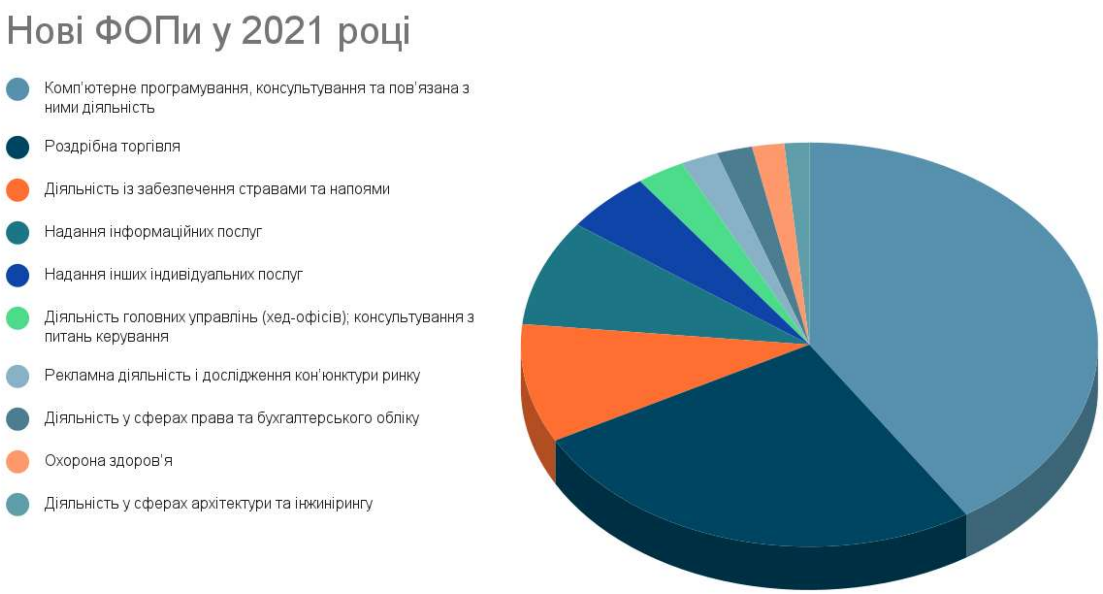


Рисунок 1.2 – Нові ФОПи у 2021 році

Це найбільший показник за останні 5 років. Незважаючи на те що, ФОПи неодноразово протестували через введення карантинних обмежень. Із 1 січня 2021 року малий бізнес у сфері торгівлі, громадського харчування й послуг має використовувати касові апарати та програмні реєстратори розрахункових операцій, що також впливає на ріст, так як це створює перешкоди для більш легкого старту. Та незважаючи на це все прогнозують ріст малого бізнесу та вихід його з тіні.

1.2. Актуальність створення додатків для керування торгівлею та складського обліку

Аналізуючи дані, які надають офіційні джерела, бачимо що, роздрібна торгівля займає значну частку та показує ріст. Товарообіг с кожним роком росте, що надає інформацію, що попит і різноманітність товарів також росте. А спираючи на стійку точку зору – пропозиціє створює попит. Тому можна впевнено вважати, що малий бізнес росте.

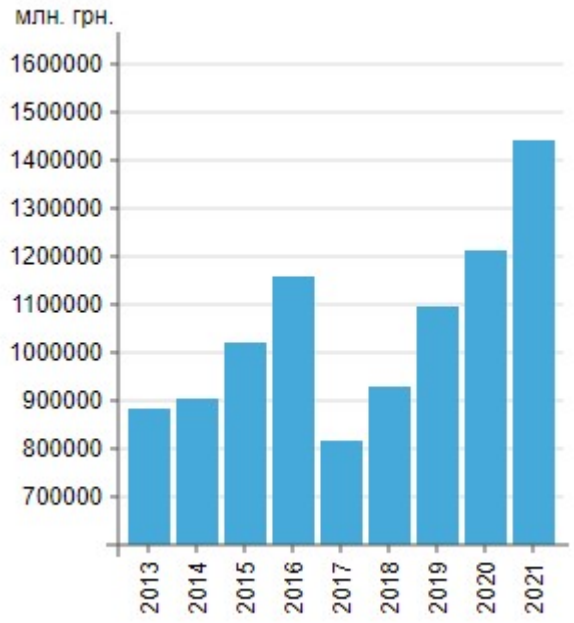


Рисунок 1.3 – Роздрібний товарообіг за роками

Значна кількість продажів іде від інтернет-торгівлі. Позиції e-commerce укріпляються на українському ринку.

Популярність електронної комерції залежить від популярності Інтернету. За різними оцінками, Інтернетом користується близько 70% населення України. Це для порівняння з 30% у 2010 році. Коли люди звикають до Інтернету, вони звикають і до послуг, які він пропонує – одна з них – покупка в Інтернеті.

За даними дослідження компанії CBR, до кінця 2020 року в Україні 10,6 млн людей регулярно купували в інтернеті – це третина населення. Йдеться про постійних клієнтів інтернет-магазинів та торгових майданчиків.

Українці виявляють інтерес до e-commerce у ролі покупців та продавців. Це опосередковано доводить статистика Google. Українські користувачі почали активно шукати, що продавати в інтернеті. За 2020 рік найпопулярнішими товарами в інтернеті були засоби індивідуального захисту, одяг та товари для дому. Очікується, що інтернет-магазинів у цих нішах побільшає у 2021 році.

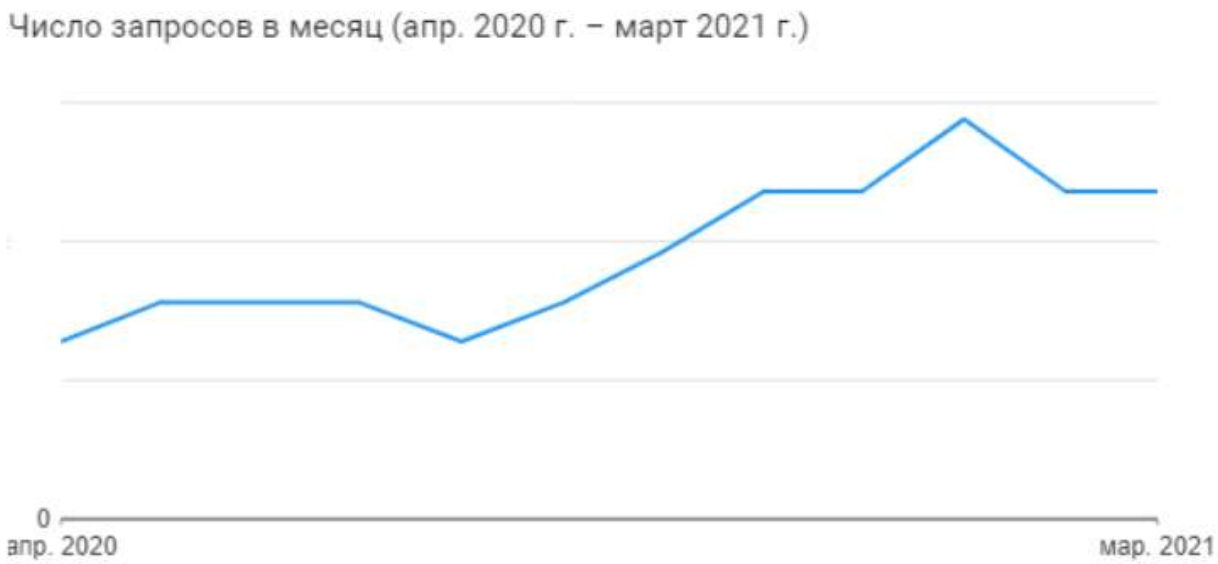


Рисунок 1.4 – Графік частотності запиту "Що продавати в інтернеті"

Експерти компанії IBM підрахували, що пандемія прискорила перехід від офлайну до інтернет-магазинів приблизно на п’ять років. Магазини, які раніше працювали лише в офлайні, почали освоювати інтернет — вони принесли до e-commerce нові ніші та розширили аудиторію покупців. Наприклад, стала

поширеною покупка продуктів в інтернеті, хоча раніше це було в основному офлайн-заняття. Активне наповнення e-commerce новими продавцями та покупцями дало результат – обсяг продажів у 2020 році в Україні зріс на 40%.

З ростом товарообігу складається навантаження на підприємців, що змушує підприємців використовувати різноманітні додатки для керування торгівлею, складського обліку, CRM-системи для автоматизації стратегій взаємодії із замовниками, зокрема для підвищення рівня продажів, оптимізації маркетингу та покращення обслуговування клієнтів шляхом збереження інформації про клієнтів.

Всі ці додатки покращають ефективність та спрощують всі бізнес процеси. Аналітика показує, що використання таких додатків, зменшує витрачання часу на обслуговування клієнта та перехід до обслуговування іншого клієнта. Також потрібно зауважити, що більшість популярних маркетплейси та платформи CMS для інтернет-магазинів, такі як: OpenCart, Joomla, InSales та інші похожі сервіси. Пропонують деякі обрізані функції з цих додатків або пропонують інтеграцію з деякими системами. Тому значна частина користувачів (власників інтернет-магазинів) переходить на використання окремих додатків для користування повним функціоналом.

Зростаючий ринок e-commerce породжує сильну конкуренцію. Покупці поряд з ціною, звертають увагу на обслуговування, стають більш вимогливими до продавців. Швидко зв'язатися з клієнтом, проконсультувати на замовлення, надіслати товар щодня. Тому використання будь-якого додатку – це мастхев для підприємців.

1.3. Огляд існуючих додатків для керування торгівлею та складського обліку

Використання програми для контролю складського обліку та торгівлі – це напевно обов’язковий додаток для підприємця. Такі додатки спрощують елементарні бізнес-процеси, які потрібно відслідковувати, але в «ручну» робити це попросту витрачання багато часу в нікуди.

Засоби для організації контролю персональної економічної активності можуть бути описані наступним переліком:

– В ручну (паперовий облік);
– Google Sheets або Excel;
– Десктоп додаток;
– Web додатки;

Спосіб в ручну включає елементарний контроль на папері. Цей підхід один із найменш ефективний, так як вимагає досить багато часу. Також цей підхід не дуже ефективний в плані аналізу даних, яких ми маємо (який товар більше покупають, скільки витрат і доходів в цілому), а аналіз даних – це важлива частина для ефективного маркетингу.

Наступний спосіб – це використання Google Sheets або Excel. Являється більш зручним способом, ніж минулий. Але для початку роботи в ньому, потрібно зробити налаштування таблиць (створення таблиць, формул і т.д.). Це може зайняти досить багато часу і більш підходить для початківців, які ще не готові витрачувати гроші для більш зручних систем.

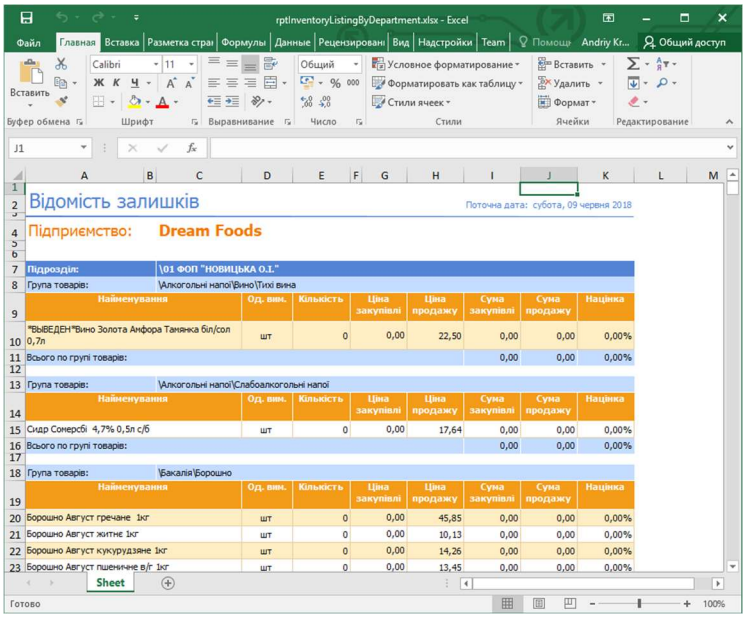


Рисунок 1.5 – Приклад таблиці Excel для ведення контролю торгівлі та складського обліку

Переваги використання Google Sheets або Excel для контролю торгівлі та складського обліку:

- Простий інтерфейс;
- Можливість автоматизувати обчислення за допомогою простих і складних формул Excel;
- Можливість розширювати функціонал за допомогою формул Excel;
- Використання графічних форм, для візуального відображення витрат/доходів;
- Можливість створювання макросів;
- Наявність засобів пошуку та фільтрації стовпців;

Та потрібно враховувати суттєві недоліки, які починають ускладнювати використання таблиць для керування торгівлею та складським обліком зі збільшенням розміру проекту, та основні недоліки можна відзначити такі:

- Створення великих таблиць вимагає досить багато часу;
- Дані постійно необхідно вводити вручну;
- Кожен може отримати доступ до вашої таблиці, так як засобів захисту майже немає;
- Для використання складних формул або написання макросів для автоматизації проблем, потрібно мати певний досвід;

Більш зручним додатками для керування торгівлею та складським обліком є десктопні додатки. З таких додатків можна відзначити найпопулярніший додаток «Дебет Плюс» - повнофункціональний програмний комплекс, що охоплює всі варіанти використання для ведення бухгалтерського обліку для малого та середнього бізнесу, а також цілком прийнятний для приватного підприємця.

Вона дозволяє керувати складом, регулювати взаємовідносини з клієнтами, реалізований модуль CRM-системи. Доступно ведення управлінського обліку, відстеження фінансових операцій, управління персоналом і розрахунок заробітної плати. Також є безкоштовна версія, з обрізаним функціоналом. Додаток наведений на рисунку 1.6.

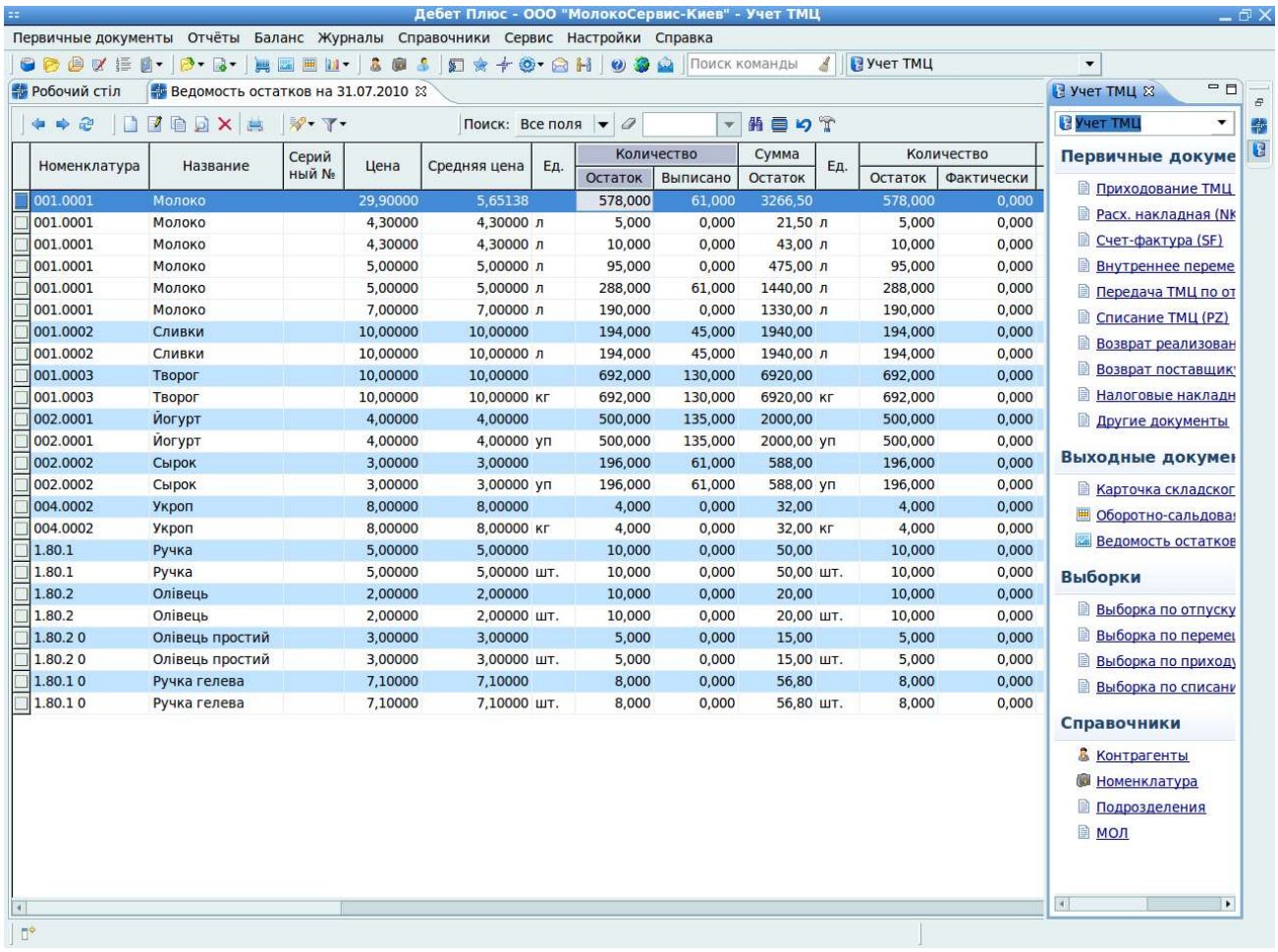


Рисунок 1.6 – Приклад додатку «Дебет Плюс»

З переваг «Дебет Плюс» можна виділити такі:

– Є безкоштовна версія, з обрізаним функціоналом;
– Є CRM модуль для роботи з клієнтами;

З недоліків «Дебет Плюс» має такі:

– Десктопний додаток. Тому неможливо працювати з інших пристроїв або віддалено;
– Застарілий інтерфейс з «лишніми кроками» для дій (створення позицій, продаж і т.д);

Десктопні програми все застарілі і ці продукти масово переходять на web-додатки, або далі продовжують роботу над клієнтськими додатками, які масово покидають користувачі.

З українських web-додатків для керування торгівлею та складського обліку будемо розглядати «ERP FOSS» та «ДІЛОВОД».

Розглянемо перший web додаток «ERP FOSS» для керування торгівлею та складським обліком. Просте та недороге рішення на ринку. Основна ціль цього додатку дати мінімальний функціонал для керування торгівлею та складським обліком.

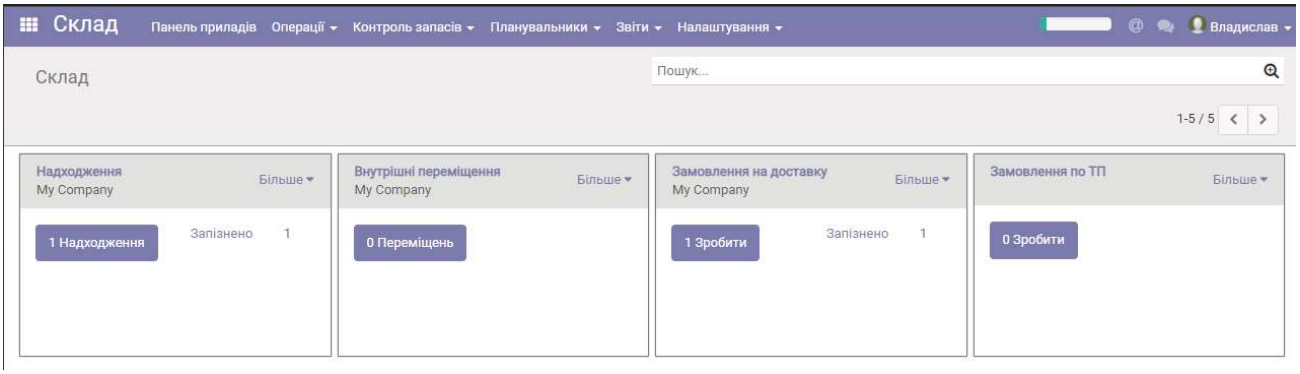


Рисунок 1.7 – Головне меню «ERP FOSS»

Як заявляють розробники, вся робота ведеться в онлайн і офлайн режимах. Програма працює з усіма одиницями вимірювання і валютами. Вона підтримує складський облік, сканери штрих-кодів. Сервіс забезпечує повну звітність і аналіз робочого процесу.

Можна вказати такі переваги цього додатку:

- Ціна – єдиний тариф, який залежить від кількості користувачів системи. Мінімальна ціна 250 грн, що скоріш за все, найдешевше рішення на ринку.
- Швидке налаштування для початку роботи.
- Інтеграція з «Prom.ua».

Інтеграцію з «Нова Пошта» не можна вказати, як перевагу. Так як, якщо ви користуєтесь «Prom.ua», то вам непотрібно ця інтеграція і більшість інтернет магазинів має плагіни для створення доставки в своїх кабінетах, так як це набагато швидше, ніж робити це в «ERP FOSS».

З недоліків можна виділити такі:

- Немає дашбордів, для аналізу продажів;

– Неможливість імпортувати таблицю з даними, яку надалі можна використовувати для різних цілей, наприклад, надавати інформацію о наявності товару, як постачальник, в .xsls , або іншому форматі;
– Не зручний інтерфейс, новачок в цій системі може загубитись та не зрозуміти з чого починати і звикнути до інтерфейсу, що займе деякий час;

Незважаючи на всі недоліки, «ERP FOSS» непогане рішення для малого бізнесу, або бізнесу с невеликою різноманітністю товарів. Цей додаток надає мінімальний функціонал, але якщо використовувати сервіси, такі як OpenCart та Joomla, або схожі сервіси за функціоналом, то «ERP FOSS» не потрібен.

Другий додаток це «PERFECTUM», який набагато функціональний ніж попередній додаток. Складне і універсальне рішення на ринку, яке може підійти як для таких галузей IT і реклама, роздрібна торгівля, послуги, фінанси, освіта та охорона здоров'я.

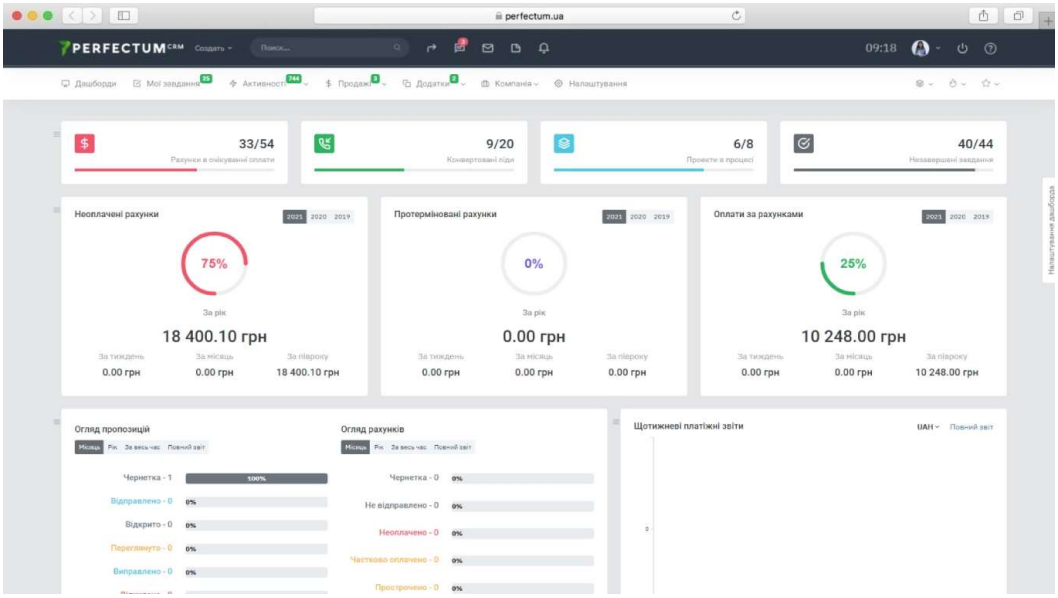


Рисунок 1.8 – Головне меню «PERFECTUM»

Додаток має обширні можливості для вирішення будь-якого бізнеса процесу. Але потрібно розуміти, що кожний модуль потрібно купляти, ціна яких від 3 тис. грн. до 20 тис. грн.. Також, як заявляють розробники, сервіс можливо «адаптувати»

під себе (змінити інтерфейс, дизайн) , а також є можливість зберігати дані на своєму сервері для безпеки.

Досліджуючи цей додаток можна виділити такі переваги:

- Можливість за додаткову плату модифікувати додаток, що надає безліч можливостей;
- Має інтеграцію з IP-телефонією «Binotel» ;
- Одноразова оплата;

Цей додаток має безліч переваг, так як розроблений універсальним рішенням. Але це і його ціна і є його головним недоліком, так як універсальне рішення надає безліч функціонала, але так як ви сплачуєте за це все, то ви переплачуєте за той функціонал який ви не використовуєте. Напевно, для вирішення цієї проблеми потрібно було ввести підписку, для того щоб вибирати тариф з функціоналом. Також, якщо обговорювати ціну і брати рішення для роздрібної торгівлі, то мінімальна ціна становить 45 тис. грн. для 5 користувачів і 130 тис. грн. для необмеженої кількості користувачів, але ще потрібно враховувати ціну модулів, то додаток вийде дорогим. Зведені результати аналізу розглянутих додатків наведено у таблиці 1.1.

Таблиця 1.1 – Результати аналізу розглянутих додатків

Назва продукту	**Переваги**	**Недоліки**
Дебет Плюс	1) Є безкоштовна версія, з обрізаним функціоналом; 2) Є CRM модуль для роботи з клієнтами;	1) Десктопний додаток. Тому неможливо працювати з інших пристроїв або віддалено; 2) Застарілий інтерфейс з «лишніми кроками» для дій (створення позицій, продаж і т.д);

Назва продукту	Переваги	Недоліки
ERP FOSS	1) Ціна – єдиний тариф, який залежить від кількості користувачів системи. 2) Швидке налаштування для початку роботи. 3) Інтеграція з «Prom.ua» та «Нова пошта».	1) Немає дашбордів, для аналізу продажів. 2) Не зручний інтерфейс, новачок в цій системі може загубитись.
PERFECTUM	1) Має інтеграцію з з IP-телефонією «Binotel». 2) Одноразова оплата. 3) Можливість за додаткову плату модифікувати додаток, що надає безліч можливостей.	1) Ціна. Додаток дуже дорогий, і підходить більш для середнього бізнесу. 2) Непотрібний функціонал. Так як система універсальна, то містить дуже багато непотрібного функціоналу для користувача, який в основному заважає.

1.4. Мова програмування C# та .NET

C# — це строго типізована об'єктно-орієнтована мова програмування. Розроблена та запущена компанією Microsoft. Яка надає сучасним розробникам гнучкість та можливості для створення програмного забезпечення, яке не тільки працюватиме сьогодні, але й буде застосовуватися протягом багатьох років. майбутнє.

Ця мова програмування дуже поширена при створенні комп’ютерних і web додатків. При розробці комп’ютерних програм для Windows важко уявити, яка мова програмування була б більш актуальною. Дуже корисно мати велику кількість

користувачів мови, тому що отримана інформація збільшується, люди зможуть допомогти вирішити труднощі, що виникли при розробці, а головне – вдосконалюється мова, створюються нові програми та інструменти.

Метою Microsoft була розробка мови програмування, яку не тільки легко вивчити, але й підтримувати сучасні функціональні можливості для всіх видів розробки програмного забезпечення.

Якщо ви подивитеся на історію мов програмування та їх особливості, кожна мова програмування була розроблена для певної мети, щоб вирішити конкретну потребу того часу. Проте мова C# була розроблена, щоб враховувати потреби бізнесу та підприємств. Мова C# була розроблена для компаній, щоб створювати всі види програмного забезпечення за допомогою однієї мови програмування.

C# надає функціональні можливості для підтримки сучасної розробки програмного забезпечення. Та підтримує потреби в розробці web-, мобільних пристроїв і програм. На синтаксис мови C# впливають C++, Java та кілька інших мов. C# також уникає складності та неструктурованих мовних функцій.

C# є open source у рамках .NET Foundation, який керується та працює незалежно від Microsoft. Специфікації мови C#, компілятори та пов’язані з ними інструменти є проектами з відкритим кодом на Github. Хоча дизайн функцій мови C# очолює Microsoft, спільнота з відкритим кодом дуже активна в розробці та вдосконаленні мови.

C# являється кросплатформеной мовою програмування. Ви можете створювати програми .NET, які можна розгорнути на платформах Windows, Linux і Mac. Програми також можна розгортати в хмарі та в контейнерах.

Говорячи про технології .NET, такі як ASP.NET, у багатьох випадках люди мають на увазі мову програмування C#. Вони тісно пов’язані між собою, але їх не слід вважати одним і тим же. Мова програмування C# була розроблена для роботи з .NET Framework, яка є набагато ширшою.

.NET — це безкоштовна кросплатформна платформа з відкритим вихідним кодом для створення сучасних, масштабованих і високопродуктивних настільних, web-, хмарних і мобільних додатків. уніфікована платформа для створення

настільних, web-, хмарних, мобільних, ігрових додатків, IoT та AI. Екосистема .NET має єдину загальну бібліотеку, середовище виконання, компілятори мов та інструменти.

Фреймворк забезпечує безпеку, сумісність з мовою програмування та загальну модель розробки для платформи Windows. Він дозволяє створювати комп'ютерні програми (наприклад, WPF), web-додатки (наприклад, ASP.NET), web-аплети (наприклад, Silverlight), використовувати бази даних (наприклад, ADO.NET), тощо за допомогою створеного API.

Мова програмування C# була створена спеціально для ASP.NET і в кінцевому підсумку написана цією мовою програмування. Технологія Active Server Page, або скорочено ASP.NET, стала новим інструментом на платформі .NET. Вона сприяє швидкому розробці web-додатків, які взаємодіють з базами даних.

1.5. Технологія WebAssembly

WebAssembly, або WASM, є форматом двійкового коду, близьким до ассемблера та незалежним від мови та платформи, оскільки WebAssembly може бути скомпільований з інших мов і може виконуватися в браузері (web-API) або на віртуальній машині. WebAssembly – це відкритий стандарт, головна мета якого полягає в тому, щоб запропонувати власну продуктивність в Інтернеті, зберігаючи сумісність із поточними екосистемами та стандартами.

WASM це не мова програмування, подібно до того як байт-код Java це не мова програмування, а результат компіляції і блок коду, що запускається. Основні цілі - крос-платформність, компактність, швидкість. Коли браузер завантажує код WebAssembly, він може швидко перетворити його на складання будь-якої машини.

WebAssembly дозволяє вам брати код C, C++, C# або Rust і компілювати його так званий модуль WebAssembly рисунок 1.6. Ви можете завантажити його у свою web-програму та викликати з JavaScript.

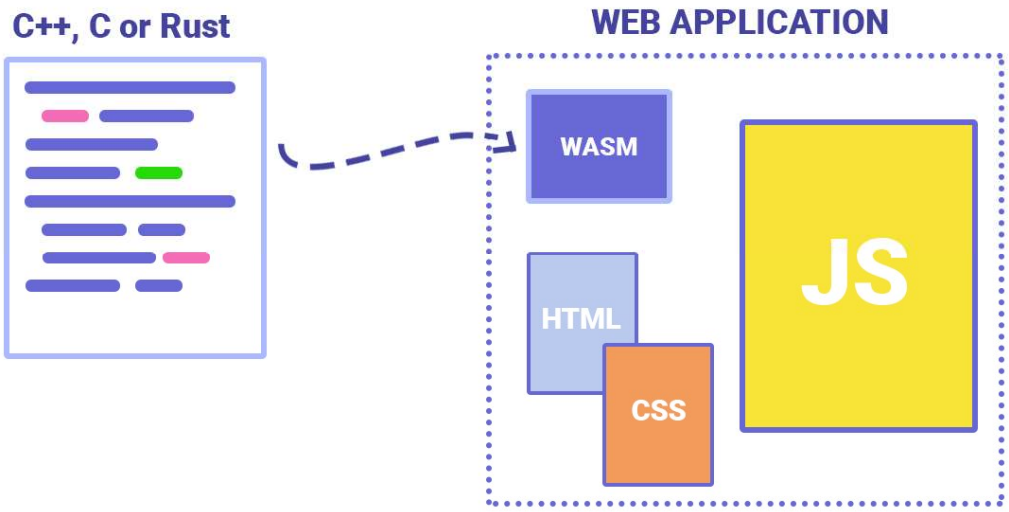


Рисунок 1.9 – Принцип роботи WASM

WebAssembly був створений для підвищення продуктивності наших програм JavaScript, оскільки він заснований на двійковому коді і використовує грубу потужність дуже продуктивних мов. Так, порівняно з чистим JavaScript, WASM швидше приблизно на 60%.

Також потрібно розуміти, що WebAssembly забезпечує двосторонній зв'язок між модулями JavaScript та WebAssembly, що дозволяє отримати вигоду від його високої продуктивності та легшого доступу до API браузера.

Це не означає, що WebAssembly замінить JavaScript у програмах, які керують API та використовують їх, але це означає, що це відмінний інструмент для підвищення продуктивності мережі. WebAssembly стає все більш важливим для web-розробки і є відмінним інструментом для покращення додатків.

Однією з головних цілей при розробці WebAssembly була переносимість. Щоб запустити програму на пристрої, вона повинна бути сумісна з архітектурою процесора пристрою та операційною системою. Це означає компіляцію вихідного коду кожної комбінації операційної системи та архітектури процесора, яку ви

хочете підтримувати. З WebAssembly є тільки один крок компіляції, і ваш додаток працюватиме в будь-якому сучасному браузері.

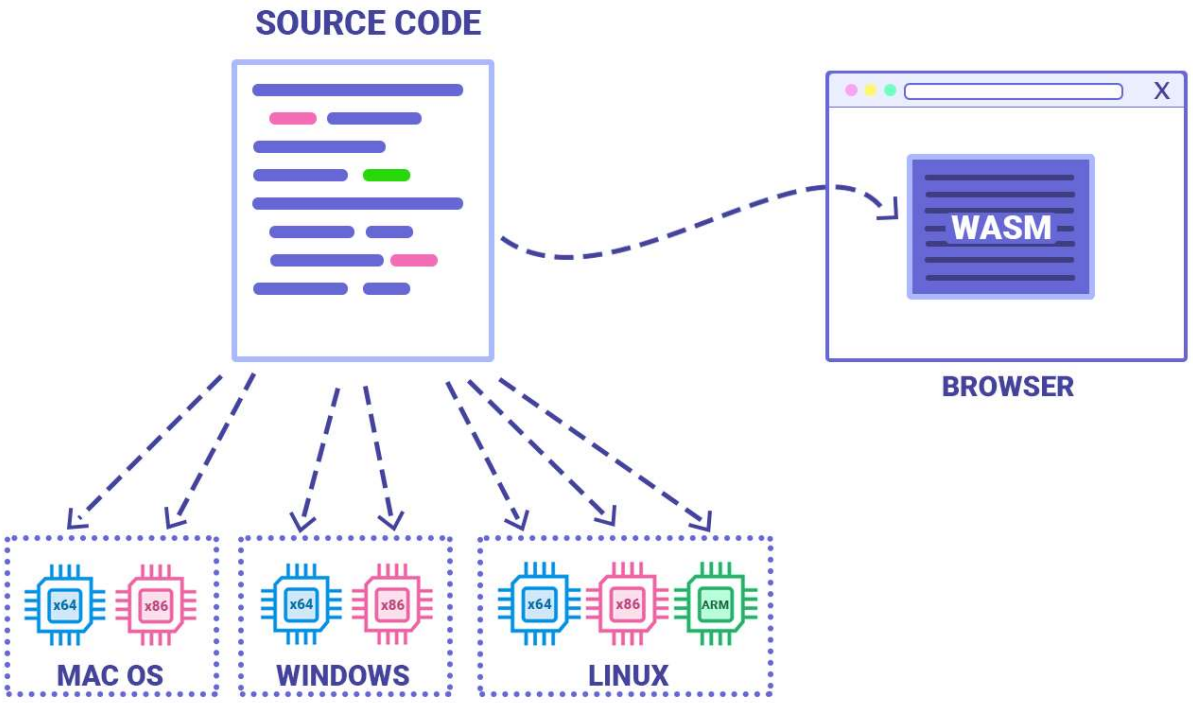


Рисунок 1.10 – Компіляція нативного коду для роботи на різних платформах у порівнянні з компіляцією в WebAssembly

Ви можете портувати в Інтернет не лише свої власні програми, але й величезну кількість бібліотек C++ та програм з відкритим вихідним кодом, які там існують. Це мова, яка підтримується практично на всіх платформах, включаючи iOS та Android. За допомогою WebAssembly його можна використовувати як спільну мову для web-додатків та мобільних додатків.

Найцікавіше в WebAssembly це те, що він забезпечує велику гнучкість при написанні для Інтернету. До цих пір JavaScript був єдиною мовою, що повністю підтримується у web-браузерах. З WebAssembly web-розробники зможуть вибирати інші мови, і більше розробників зможуть писати код для Інтернету. JavaScript, як і раніше, буде найкращим вибором для більшості випадків використання, але тепер

буде можливість час від часу перемикатися на спеціалізовану мову, коли вам дійсно потрібне прискорення.

1.6. Огляд фреймворка Blazor

Blazor - це нове клієнтське середовище інтерфейсу користувача від команди ASP.NET. Його великою перевагою є можливість створення багатого web-інтерфейсу з використанням HTML, CSS і C# замість JavaScript - те, про що мріяли багато розробників.

Але Blazor вже почав демонструвати, що у нього є потенціал як високоефективна і продуктивна модель програмування за межами його початкового дизайну — як прямий конкурент фреймворкам одно сторінкових додатків JavaScript (SPA).

Для розробки за сервера ми використовуємо такі мови програмування, як C#, Java, PHP тощо. буд. Це мови програмування за сервера. Для розробки на стороні клієнта ми використовуємо фреймворки JavaScript, такі як Angular, React, Vue і т. д. Немає жодних сумнівів у тому, що донедавна ці фреймворки JavaScript домінували у розробці на стороні клієнта.

Щоб залишатися в бізнесі як розробник і залишатися конкурентоспроможними, ми неминуче вивчаємо мову програмування як серверної, так клієнтської сторони. Але питання в тому, навіщо нам вивчати та використовувати 2 різні набори мов програмування та фреймворків.

Рисунок 1.11 – Сучасна розробка web-додатків

З Blazor ми тепер можемо створювати інтерактивні web-інтерфейси, використовуючи C# замість JavaScript. Код C# може виконуватися як у сервері, і у клієнтському браузері. Це означає, що існуючі розробники .NET можуть повторно використати свої навички роботи з C#, а не вивчати нові фреймворки JavaScript.

Рисунок 1.12 – Розробка web-додатків з Blazor

По суті, у Blazor є поділ між тим, як він обчислює зміни інтерфейсу користувача (модель програми/компонента) і тим, як ці зміни застосовуються (рендерер). Це відрізняє Blazor від інших фреймворків інтерфейсу користувача, таких як Angular або ReactJS/React Native, які можуть створювати тільки інтерфейси користувача на основі web-технологій. Використовуючи різні засоби візуалізації, Blazor може створювати як web-інтерфейси, а й власні мобільні інтерфейси. Blazor пропонує 2 модели web-додатків:

- Blazor WebAssembly;
- Blazor Server;

Blazor WebAssembly дозволяє створювати інтерактивні одно сторінкові програми, які запускаються на браузері користувача за допомогою технології WebAssembly. При побудові та запуску програми Blazor WebAssembly файли з кодом C# та Razor компілюються у збірки .NET. Потім Blazor WebAssembly (а якщо точніше скрипт blazor.webassembly.js) завантажує середовище виконання. За

допомогою взаємодії з JavaScript фреймворк Blazor WebAssembly може звертатися до DOM та API браузера.

Однією з переваг Blazor WebAssembly є те, що він може оптимізувати завантаження. Зокрема, при публікації програми невикористовуваний код забирається лінкером (компонувальником) IL (Intermediate Language).

При цьому Blazor WebAssembly не залежить від сервера. За великим рахунком, нам може бути достатньо статичного сервера, на якому розміщені всі файли програми. Всі необхідні файли завантажуються браузером, і після завантаження файлів програма працює повністю на стороні браузера і не залежить від сервера.

Blazor Server — у цій моделі програма виконується на сервері з програми ASP.NET Core. Між клієнтом та сервером встановлюється з'єднання SignalR. Коли на клієнті відбувається подія, наприклад, натискання кнопки, інформація про подію відправляється на сервер через підключення SignalR. Сервер обробляє подію, і згенерованого HTML обчислюється diff (різниця).

Весь HTML-код не надсилається назад клієнту, а лише відмінності, які надсилаються клієнту через встановлене з'єднання SignalR. Потім браузер оновлює інтерфейс користувача. Blazor використовує архітектуру одно сторінкових програм, яка динамічно перезаписує ту саму сторінку у відповідь на дії користувача. Оскільки для оновлення інтерфейсу користувача застосовується тільки різниця.

Ключовим елементом програми Blazor є компоненти. Хто працював з фреймворками клієнтської сторони, такими як Angular, React, VueJS, стикався з компонентами, які по суті структурують додаток. У Blazor застосовується подібна концепція.

Тут компонент представляє елемент інтерфейсу, наприклад, певний зміст, меню, діалогове вікно, форма введення даних. Компоненти визначають логіку рендерингу елементів інтерфейсу, а також логіку обробки введення користувача.

Компоненти можуть бути вкладені в інші компоненти. Компоненти можна повторно використовувати в інших проектах та переносити у вигляді бібліотеки

класів Razor. Зазвичай клас компонента міститься у файлі з розширенням .razor, а їх визначення застосовується синтаксис Razor, що дозволяє об'єднати розмітку HTML з кодом на C#.

1.7. Архітектура web додатків на ASP.NET Core Blazor

1.7.1. Використання ASP.NET

ASP.NET — це платформа web-розробки, яка надає модель програмування, комплексну програмну інфраструктуру та різноманітні служби, необхідні для створення надійних web-програм для ПК та мобільних пристроїв.

ASP.NET працює поверх протоколу HTTP і використовує команди та політики HTTP для встановлення двостороннього зв'язку та взаємодії між браузером та сервером.

ASP.NET є частиною платформи Microsoft. Програми ASP.NET являють собою скомпільовані коди, написані з використанням компонентів або об'єктів, що розширюються і багаторазово використовуються в .NET Framework. Ці коди можуть використовувати всю ієрархію класів у .NET Framework. Програми ASP.NET також можуть бути написані мовами .NET. До них відносяться C#, VB.NET та інші.

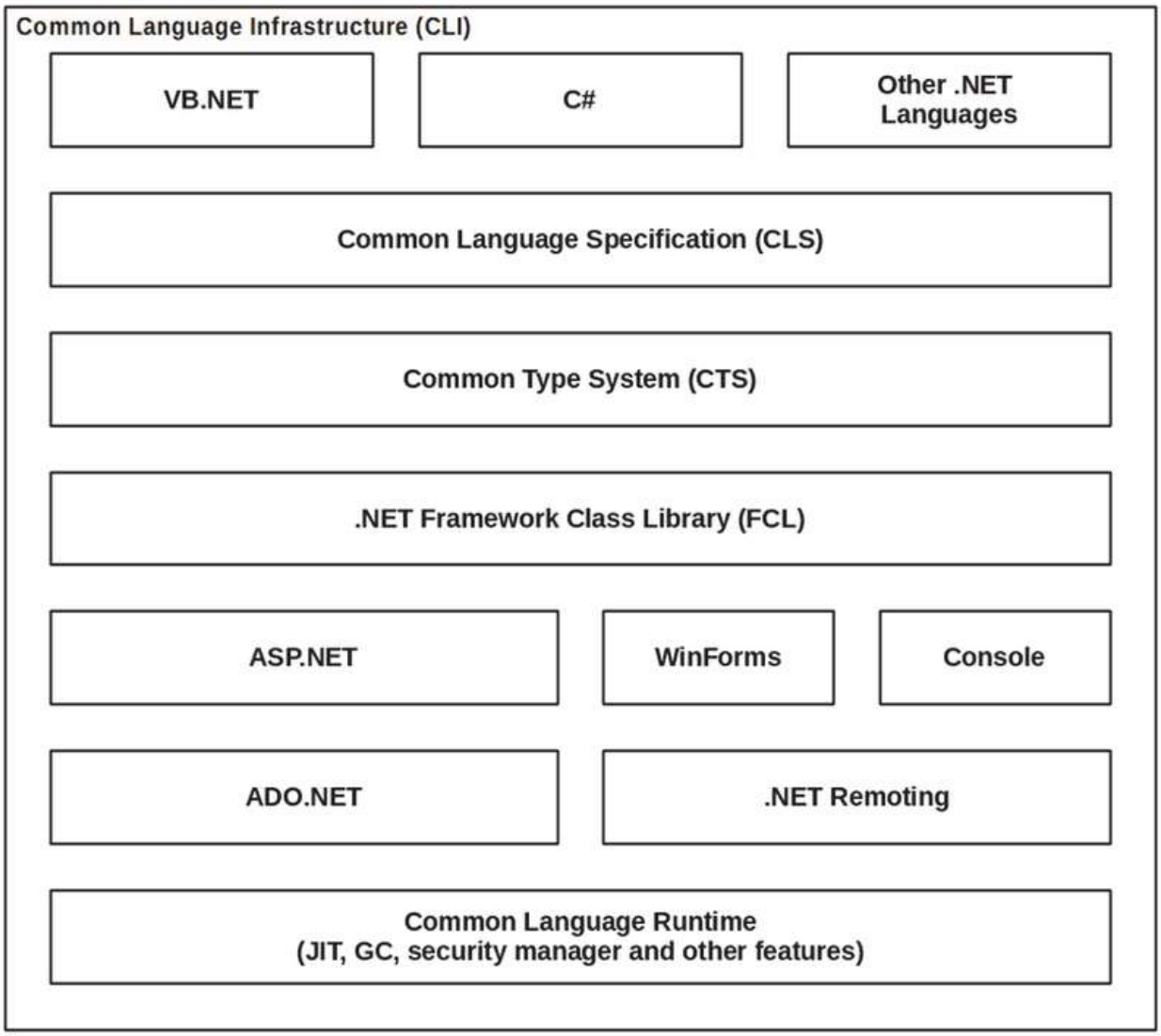


Рисунок 1.13 – Діаграма архітектури ASP.NET та її компоненти

Архітектура .NET Framework заснована на наступних ключових компонентах:

- Для .NET Framework існує багато мов. Це VB.net та C#. Їх можна використовувати для розробки web-програм.
- .NET Framework містить набір стандартних бібліотек класів. Найпоширенішою бібліотекою, яка використовується для web-застосунків у .net, є web-бібліотека. У бібліотеці є всі необхідні компоненти, які використовуються для розробки web-додатків .NET.
- Common Language Runtime – Common Language Infrastructure або CLI – це платформа. Програми .NET виконуються на цій платформі. CLR

використовується для виконання ключових дій. Дії включають обробку винятків та складання сміття.

Web-сайти та програми, створені за допомогою ASP.NET, можуть бути швидше та ефективнішими, ніж, наприклад, web-сайти, створені за допомогою PHP. Програми ASP.NET компілюються, що означає, що код транслюється в об'єктний код, який потім виконується. Цей процес компіляції займає невелику кількість часу, але відбувається лише один раз. Після компіляції код може виконуватися платформою .NET знову і знову дуже швидко.

ASP.NET – це технологія, що працює на платформі .NET, яка містить усі функції, пов'язані з Інтернетом. Платформа. NET складається з об'єктно-орієнтованої ієрархії. Web-програма ASP.NET складається зі сторінок. Коли користувач запитує сторінку ASP.NET, IIS делегує обробку сторінки в системі виконання ASP.NET.

Середовище виконання ASP.NET перетворює сторінку .aspx на екземпляр класу, який успадковується від сторінки базового класу платформи .Net. Отже, кожна сторінка ASP.NET є об'єктом, і її компоненти, т. е. елементи управління за сервера, також є об'єктами.

Програми ASP.NET можуть використовувати всі популярні бази даних, включаючи Microsoft SQL Server, MySQL, MariaDB, Postgres, MongoDB та CouchDB.

1.7.2. Синтаксис розмітки Razor

Razor — це інтелектуальний механізм коду для динамічних web-сторінок ASP.NET. Він має простий та інтуїтивно зрозумілий синтаксис для вбудовування коду у web-сторінки. Razor — це механізм перегляду, яка підтримує .NET Framework, .NET Core в ASP.NET і призначена для створення web-додатків.

Механізм Razor надає вам усі можливості ASP.NET, але у спрощеному синтаксисі, який є простим для новачків і підвищує продуктивність для професіоналів. Razor дозволяє писати код без великої кількості відкривальних і закриваючих тегів по всьому шаблону, що робить процес розробки дуже швидким.

Механізм Razor використовує мінімальну кількість символів для вказівки меж коду, і блоки сервера не потрібно явно позначати в коді HTML.

Використовуючи механізм візуалізації Razor, ви можете інтуїтивно й легко створювати складні композиції коду, які координуються з розміткою HTML. Коли сервер читає сторінку, двигун Razor обробляє код сторінки, перш ніж сервер надсилає отримані дані браузеру. Web-сторінка, створена серверним кодом і завантажена в браузер, нічим не відрізняється від статичного вмісту HTML. Web-сторінки ASP.NET із синтаксисом Razor мають певні розширення .cshtml (за допомогою Razor C#) і .vbhtml (з використанням Razor Visual Basic).

Серверний код може створювати динамічний web-контент. Користувач щось робить у браузері, браузер надсилає запит, а сервер його обробляє. На основі даних браузера сервер інтерпретує відповідний файл Razor для створення нової сторінки або частини HTML. Отриманий HTML надсилається в браузер і відображається користувачеві. Обсяг коду, необхідний для опису однієї сторінки на Razor, у кілька разів менше, ніж той самий обсяг на HTML. Razor виконує деякі завдання, які зазвичай доводиться писати на JavaScript (для цього потрібен експерт зі знанням 2 мов програмування). Razor може отримати доступ до баз даних і виконувати прості (або, швидше, складні) маніпуляції з даними для їх відображення.

Razor — це C# у HTML із майже таким же синтаксисом, що й чистий C#:

- Блоки коду Razor починаються з: @...{;
- Вбудовані вирази (змінні та функції) починаються з символу @;
- Оголошувати змінні за допомогою ключового слова var (але ви можете оголошувати змінні строго типізованого типу);
- Файли Razor мають розширення .cshtml (C# змішано з HTML) ;

1.7.3. Бібліотека компонентів MudBlazor

MudBlazor – це бібліотека компонентів Blazor, що містить близько 70 компонентів. Усі компоненти та функції доступні в бібліотеці, тому вам не потрібно додавати будь-які залежності. Логіка, що лежить в основі цих компонентів, на 99,9% написана на C#, тому, як розробник .NET ви можете легко

адаптувати її під свої потреби або просто налагоджувати. Крім того, ця бібліотека має повний контроль над CSS, тому знання CSS не потрібні для створення добре спроектованих web-додатків

Майже всі компоненти використовують тільки C# (без Javascript, за винятком випадків, коли це необхідно). Бачення MudBlazor полягає в тому, щоб зробити його чистим, простим і мати сучасний дизайн, що настроюється. Ніякого Javascript, тільки Blazor та CSS. Команда задокументувала бібліотеку у дуже зрозумілій формі.

MudBlazor повністю реактивний, тобто його можна використовувати як у браузері, так і в настільних та мобільних додатках, тому створювати додатки для багатьох систем дуже просто.

MudBlazor – одна з бібліотек компонентів Blazor. Є й інші, такі як Radzen або Syncfusion і т. д., але нині найбільший інтерес суспільства має MudBlazor. Мабуть, тому, що має найкращу документацію і найшвидшу продуктивність. І це наводить до того він набагато швидше розвивається, ніж його конкеренти.

1.7.4. Використання ORM-інструмент Entity Framework Core для роботи з СУБД

Більшість сучасних програм ASP.NET Core використовують Entity Framework Core. Entity Framework Core — це технологія доступу до бази даних від Microsoft. Він дозволяє вам взаємодіяти з базою даних, використовуючи сутності (тобто класи та об’єкти NET) замість таблиць бази даних. Це найвідоміший і найфункціональніший інструмент ORM на C#. ORM — це об’єктно-реляційне відображення — зіставлення даних у реальні об’єкти.

Наприклад, якщо розробник працює безпосередньо з базою даних, програміст повинен враховувати підключення, підготовку SQL і параметри SQL, а також спосіб надсилання запитів і транзакцій. З Entity Framework Core все це робиться автоматично — розробники працюють безпосередньо з класами NET.

ORM ділиться на декілька типів:

– Code First. Це означає, що код написаний на C#, а потім за допомогою цього коду створюється база даних. Для цього підходу важливо визначити клас моделі або сутності, яка буде зберігатися в базі даних, описати її як модель у класі C# і написати клас контексту, який працюватиме з базою даних, що використовується. Програмісти на C# найчастіше використовують підхід Code First;
– Database-First. Потрібні гарні знання SQL, але не обов'язково C# у цьому випадку. Спочатку створіть базу даних, а потім створіть модель бази даних EDMX. Цей XML у файлі .edmx містить інформацію про структуру бази даних, модель даних та її відображення. Visual Studio дозволяє використовувати .edmx;
– Model-First. Архітектори часто використовують його, оскільки цей підхід не вимагає знання синтаксису SQL або C#. У цьому випадку спочатку створюється модель графіка EDMX, тоді як класи моделі C# створюються за лаштунками, а потім база даних створюється з графіка EDMX;

Як технологія ORM, відмінною рисою Entity Framework Core є використання запитів LINQ для вилучення даних з бази даних. Використовуючи LINQ, ми створюємо різноманітні запити для вибору об'єктів, у тому числі пов'язаних з різними асоціаціями. Entity Framework перетворює вирази LINQ у вирази, які може зрозуміти конкретна база даних.

LINQ нерозривно пов'язаний з Entity Framework в NET. LINQ розшифровується як Language Integrated Query або Intra-Language Query, яка являє собою технологію, що являє собою набір функцій в NET, що дозволяють писати структуровані запити до бази даних.

Для роботи з Entity Framework Core використовується технологія LINQ to Entities. LINQ витягує дані з бази даних за допомогою виразів C#, подібних до SQL. Будь-яка реляційна база даних працює за допомогою запитів SQL, а Entity

Framework Core перекладає вирази LINQ to Entities у запити SQL, які може зрозуміти база даних, що використовується.

1.8. Складання завдань дослідження

У ході досліджень було визначено, що потрібно проаналізувати аналоги, які мають схожий функціонал. Для проектування системи потрібно дослідити вибрані конкретні інструменти та засоби реалізації.

Тому для ефективної побудови web-додатку, були поставлені завдання на дослідження вибраних технологій та інструментів, що дозволять реалізувати розроблену концепцію, а саме:

- Дослідити інструменти побудови web-додатків за допомогою ASP.Core Blazor;
- Дослідити ефективність Entity Framework Core для роботи з даними;
- Дослідити структуру даних системи та підібрати БД;

2. РОЗРОБКА СТРУКТУРИ WEB-ДОДАТКУ ДЛЯ КЕРУВАННЯ ТОРГІВЛЕЮ ТА СКЛАДСЬКОГО ОБЛІКУ

2.1. Завдання web додатку для керування торгівлею та складського обліку

Контроль бізнес процесів компанії – це досить клопітка робота, яка потребує терпіння із-за одноманітності роботи. Вирішення цієї проблеми це системи, яка автоматизують бізнес-процеси компанії. Основна мета системи підвищити загальну продуктивність бізнесу за рахунок зменшення кількості «ручних» операцій, збору та накопичення даних, а також упорядкування бізнес-процесів всередині компанії.

У міру зростання їх бізнесу багато компаній почали розуміти, що їм потрібна система. Навіть у малому бізнесі цей інструмент нероздільний, а середній бізнес все частіше використовує такі інструменти. Такі додатки є передусім інформаційною системою, яка дозволяє зберігати та обробляти більшість критичних даних для роботи вашої компанії.

Основна перевага додатків це покращення ефективності та спрощування всі бізнес процесів. Аналітика показує, що використання таких додатків, зменшує витрачання часу на обслуговування клієнта та перехід до обслуговування іншого клієнта.

Проектована система дозволить виконувати такі задачі, пов’язані з керуванням торгівлею та складським обліком:

- Створення нових позицій товарів та можливість редагування;
- Створення брендів та категорій товарів для розподілу, та подальша можливість редагування;
- Створення закупок;
- Створення продаж;
- Відображення продаж та закупок;
- Візуальне інформування по затратам за доходам;

– Створення прайс-листів;

Мета роботи – розробка web додатку для керування торгівлею та складського обліку на платформі ASP.NET Core Blazor

2.2. Моделювання об’єкту проектування

2.2.1. Діаграма прецедентів системи

Діаграми прецедентів системи є основною формою системних/програмних вимог для нових програм. Використання опції визначає очікувану поведінку (що), а не точний спосіб її реалізації (як). Коли варіанти використання визначені, вони можуть бути представлені в текстових і візуальних представленнях (тобто схематичних варіантах використання).

Ключова концепція використання моделювання полягає в тому, що воно допомагає нам проектувати з точки зору кінцевого користувача. Це ефективний метод роботи з користувачами та замовниками.

Робота з діаграмою прецедентів зазвичай проста. Вона не показує деталі використання параметрів:

– Вона демонструє лише деякі відносини між використанням, між діючими акторами;
– Вона не показує послідовність кроків, зроблених для досягнення цілей кожного випадку використання;

Актор — це набір логічно пов’язаних ролей, які виконують під час взаємодії з прецедентом або сутністю (системою, підсистемою чи класом). Актором може бути користувачем або іншою системою, підсистема чи клас, вони ніщо. Графічні актори представлені «людьми».

Прецедент (Use Case, випадок використання) – опис набору наступних подій (включаючи варіанти), які можуть бути виконані системою, що призведе до спостережуваного актором результату.

Прецеденти – це поведінка сутностей, що описують взаємодію між акторами та системою. Прецедент показує не те, «як» були досягнуті певні результати, а

лише «що було виконано». Прецедент позначається дуже просто - у вигляді еліпса з його назвою всередині.

При аналізі та проектуванні варіантів діаграми може дати вам уявлення про те, які результати хочете отримати користувач.

Метою даного проекту є створення web-додатку, яка буде зберігати інформацію про товари, закупки, продажі, створення прайс-листів і відображенням доходів та розходів, які будуть формуватися від продаж за закупок.

Користувачем даної системи будуть менеджери/керівник малого та середнього бізнесу. На Рисунку 2.1 показані основні можливості, які будуть користувачу.

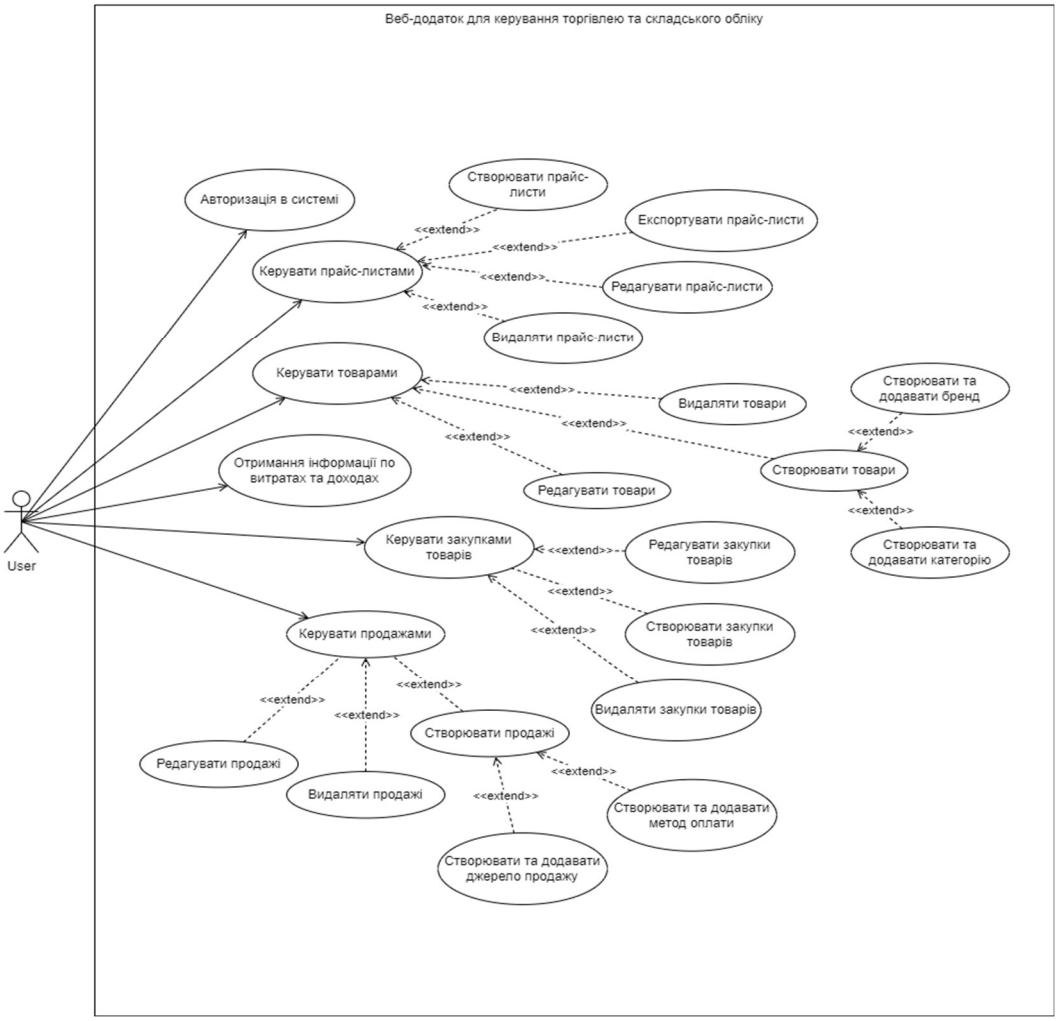


Рисунок 2.1 – UML Діаграма прецедентів системи

2.2.2. Діаграма діяльності

Діаграма діяльності, як і діаграма стану, він відображає динамічні аспекти поведінки системи. Діаграма використовується для відображення послідовності дій.

Діаграма діяльності показують робочий процес від початку до кінця, докладно описуючи багато способів, якими рішення існують у послідовності подій, що містяться в дії. Їх можна використовувати для деталізації ситуацій, коли під час певних операцій можуть відбуватися паралельні операції.

Діаграми діяльності корисні для бізнес-моделювання, вони використовуються для деталізації бізнес-процесів. На Рисунку 2.2 показаний основний робочий процес web додатку.

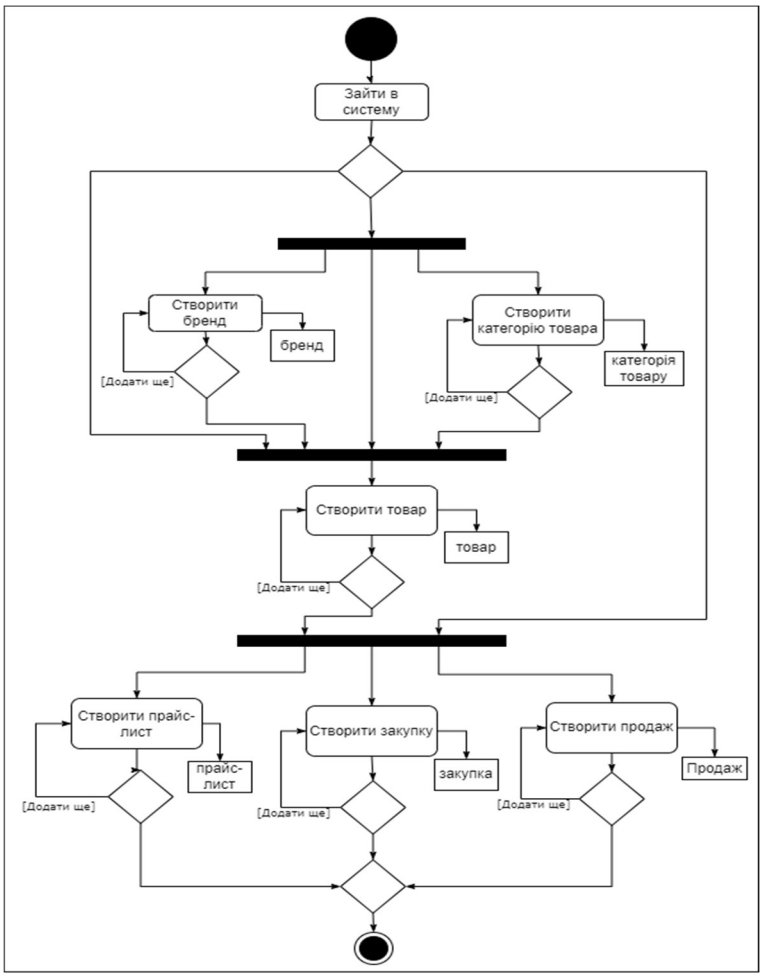


Рисунок 2.2 – UML Діаграма діяльності

2.2.3. Структура даних в системі

Оскільки ми використовуємо реляційну СУБД SQL Server, структура нашої бази даних скрадатиметься з таблиць, кожна з яких складається з рядків та стовпців. Кожен рядок зберігає окремий об'єкт, а стовпці розміщуються атрибути цього об'єкта.

Первинний ключ використовується для ідентифікації кожного рядка в таблиці. Один або кілька стовпців можуть виступати в якості первинного ключа. Використовуючи первинний ключ, ми можемо посилатися на певний рядок таблиці. Тому два рядки не можуть мати однаковий первинний ключ.

За допомогою ключів одна таблиця може бути пов’язана з іншою, тобто між двома таблицями можна організувати зв’язки. А саму таблицю можна представити як відношення («відношення»).

Розглянемо приклад структури даних, які використовуються у системі:

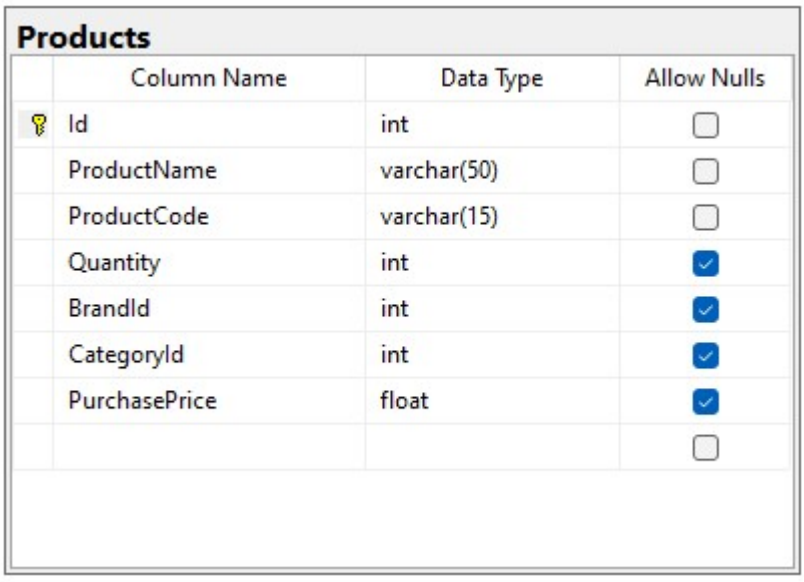


Рисунок 2.3 – Таблиця бази даних

На рисунку 2.3 показаний приклад таблиці в системі. Основним елементом бази даних є таблиця. На ній ми можемо побачити рядки, з яких вона складається та їх типів даних. Також на цій таблиці ми можемо побачити, які рядки можуть

приймати не «null» значення. Основне в таблиці - це первинний ключ (Primary key), за допомогою нього ми зв’язуємо одну таблицю з іншою.

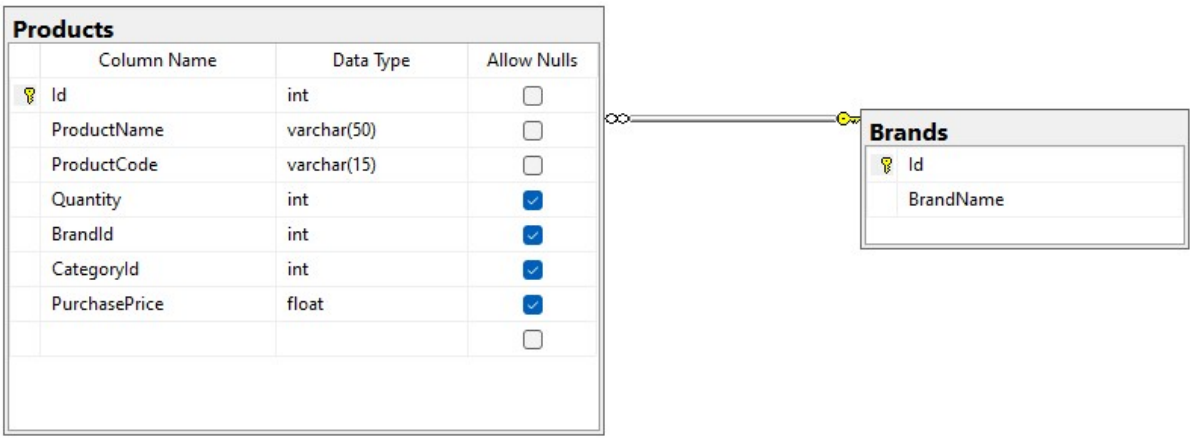


Рисунок 2.4 – Зв’язок між двома таблицями

Розглядаючи далі, на рисунку 2.4 ми можемо побачити, як виглядає зв’язок між двома таблицями. Тобто ми бачимо, що таблиця Products зв’язана з таблицею Brands. І в нашому випадку ми посилаємся рядком BrandId на первинний ключ таблиці Brands. З цього випадку BrandId називається Foreign key, або зовнішній ключ.

Зовнішні ключі використовуються для встановлення зв'язків між таблицями. Зовнішній ключ встановлюється для стовпця залежної таблиці, підпорядкованої таблиці та вказує на один із стовпців головної таблиці. Хоча зовнішні ключі зазвичай вказують на первинні ключі у пов’язаній первинній таблиці, це не обов’язково є обов’язковою умовою. Зовнішній ключ також може вказувати на інший стовпець з унікальними значеннями.

2.2.4. Робота з даними за допомогою EF Core та SQL Server

Так як в додатку ми використовуємо Entity Framework Core, як ORM-інструмент для роботи з даними. В нас є 3 підходи для роботи з даними:

- Code-First;
- Database-First;

– Model-First;

Ми будемо використовувати підхід Code-First, так як він не вимагає від нас роботи з SQL запитами. Цей підхід надає нам можливість створити таблицю в базі даних від класу, який ми створимо. Тобто принцип цього підходу, те що ми не відволікаємось на SQL запити, а створюємо таблицю від створених класів.

Рисунок 2.4 – Створення таблиці вибраного класу

На рисунку 2.4 ми бачимо, як відбувається створення таблиці. За допомогою EF Core. Це відбувається із за того, що головною особливостею EF Core – це використання LINQ, який надає нам можливость створювати запити до джерела даних. Існує декілька різновидів LINQ:

– LINQ to Objects;
– LINQ to Entities;
– LINQ to XML;
– LINQ to DataSet;
– Parallel LINQ (PLINQ);

В нашому випадку ми використовуємо LINQ to Entities з допомогою, якої ми звертаємся до бази даних за допомогою Entity Framework Core.

```
modelBuilder.Entity<Product>().HasData(
    new Product { ProductId = 1, ProductName = "Monitor", ProductCode = "01013", Price = 1000, Quantity = 1 },
    new Product { ProductId = 2, ProductName = "Mouse", ProductCode = "01012",Price = 5000, Quantity = 4,  }
);
```

Рисунок 2.5 – Добавлення даних таблиці вибраного класу

Після створення таблиці ми можемо заповнювати таблицю напряму з коду, без використання SQL – запитів. Це ми можемо побачити на Рисунку 2.5.

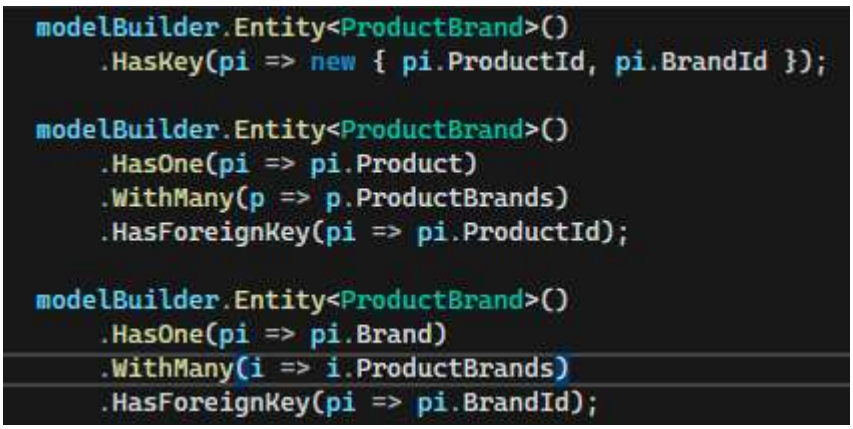


Рисунок 2.5 – Встановлення зв’язків між таблицями

Далі ми повинні створити зв’язки для таблиць. Для того щоб у нас створились первині ключи (Primary key) і зовнішні ключи (Foreign key) таблиць. Крім створення ключів, ми бачимо створення сутностей один до багатьох. Всього існує 3 сутності:

- Один до багатьох. Зв’язок «один до багатьох» виникає, коли первинний ключ однієї таблиці стає зовнішнім ключем іншою таблицею;
- Один до одного. Зв'язок між інформацією в двох таблицях, при якій вказується запис у кожній таблиці з'являється тільки один раз;
- Багато до багатьох. Зв'язок «багато до багатьох» виникає між сутностями, коли зв'язок «один до багатьох» між ними працює в обох напрямках;

Після всіх цих маніпуляцій ми маємо вибір використовувати локальну базу даних, або ж підключити до однієї БД з наведених:

- SQL Server;
- SQLite;
- PostgreSQL;

На етапі розробці найкращий варіант це робота з локальною базою даних. А на кінцевому етапі підключати БД. Для даного додатка ми підключили SQL Server.

SQL Server має певні недоліки такі як: він не підтримує enums і те що деякі типи даних C# SQL Server не підтримує. І для існуючих типів даних C# є еквіваленти в середовищі даних Microsoft SQL Server.

2.3. Структура системи

2.3.1. Структура ASP.NET Core Blazor

Як вже згадувалося у розділі 1.7, ми використовуємо платформу ASP.NET Core Blazor, для того щоб збудувати web-додаток.

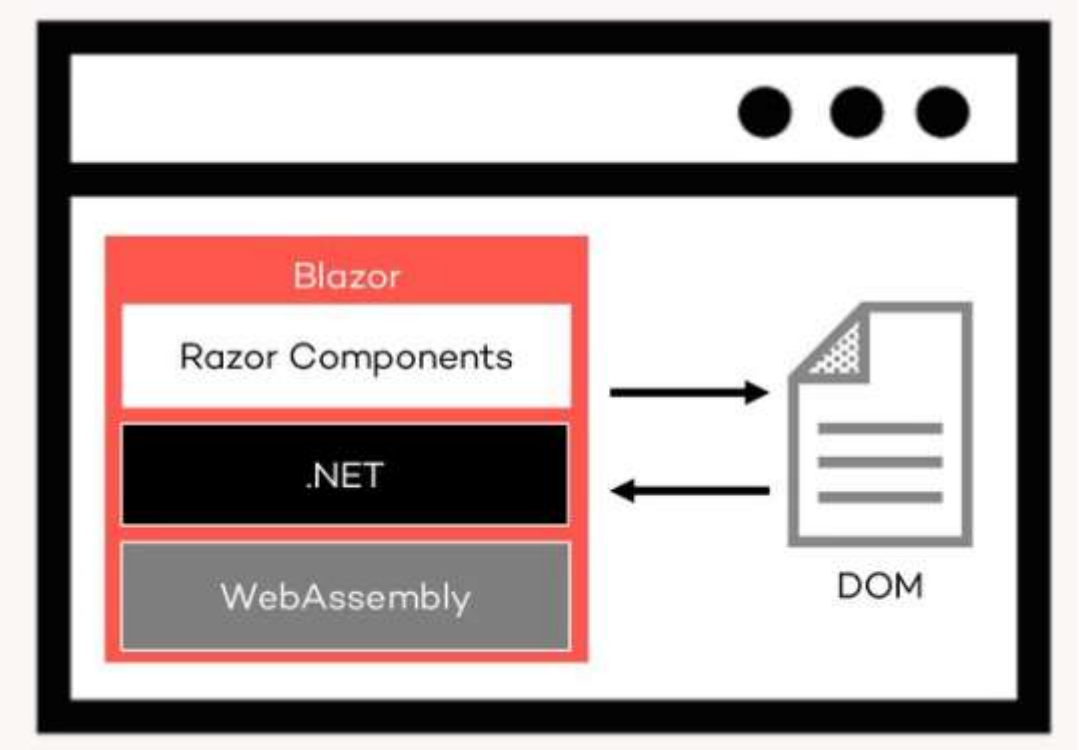


Рисунок 2.6 – Модель розміщення ASP.NET Core Blazor

На рисунку 2.6 ми бачимо, що клієнт Blazor WebAssembly (Wasm) працює у браузері. Коли користувач відкриває web-сторінку або програму, завантажується весь код, пов’язаний з клієнтською логікою. Це означає, що всі залежності також будуть завантажені. Таким чином, час виконання буде залежати від запуску програми. Після того, як ми всі завантажимо, не буде проблем, якщо ми від’єднаємося. Оскільки модель розміщення Blazor WebAssembly дозволяє нам продовжувати використовувати програму в автономному режимі, ми можемо синхронізувати зміни пізніше.

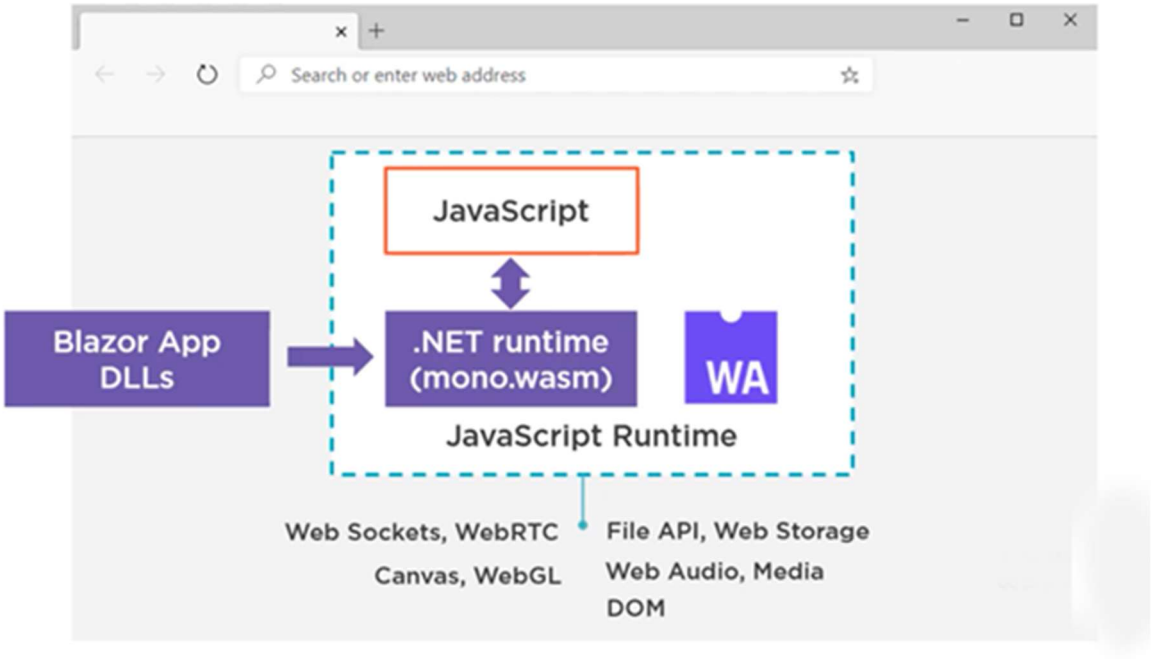


Рисунок 2.7 – Стандартна архітектура ASP.NET Core Blazor

Коли ви створюєте програму Blazor, вона створює збірки. Ці DLL є стандартними DLL .NET, які можна використовувати в будь-якій іншій сумісній програмі, що означає, що ви можете створити бібліотеку компонентів Blazor і використовувати її повторно. Ці DLL — це все, що складає вашу програму та її залежності.

У браузері ці DLL можуть запускати середовище виконання .NET. Це середовище виконання .NET є версією, скомпільованою з байт-кодом WebAssembly і названою розширенням файлу WebAssembly .wasm. Це середовище виконання .NET працює на WebAssembly у браузері. У web-браузері WebAssembly працює в ізольованому середовищі виконання JavaScript, як звичайний код JavaScript. Тут він запускає середовище виконання .NET, таке як mono.wasm, яке запускає DLL програми Blazor.

Оскільки все це виконується в середовищі виконання JavaScript, ваша програма може отримати доступ до таких функцій браузера, як WebSockets, API файлів і DOM. Це використовується для доступу до DOM для створення та оновлення інтерфейсу користувача на екрані. Крім того, він працює в середовищі

JavaScript, і код вашої програми може взаємодіяти з кодом JavaScript. Ви можете викликати JavaScript із C#, а JavaScript також може викликати ваш код C#.

2.3.2. Clean Architecture для ASP.NET Core Blazor

Вся ідея цієї архітектури полягає в тому, щоб дозволити основній частині, що складається з повної бізнес-логіки та прикладних об'єктів, бути адаптивною і досить гнучкою, щоб справлятися з технологіями та інтерфейсами, що змінюються. Крім того, основна програма залишається незмінною і не залежить від рівнів уявлення, інфраструктури та баз даних.

У цих швидко розвинених технологіях JavaScript, web-фреймворки, база даних і зовнішні частини стають абсолютними або оновлюються, використовуючи цю чисту архітектуру, ми можемо замінити ці елементи з мінімальними зусиллями. Ця архітектура заощаджує величезну кількість часу, оскільки бізнес-логіка та сутність основної програми не залежать від фреймворку, презентацій, баз даних та зовнішніх частин. Отже, архітектура є стійкою та слабко пов'язана основною бізнес-логікою та сутністю з рівнем уявлення чи структурою.

Основна ідея чистої архітектури полягає в тому, щоб зробити рішення адаптивним, зберегти основні варіанти використання бізнес-логіки або логіки програм незалежними від зовнішнього інтерфейсу та зовнішніх середовищ. Підсумовуючи, ми можемо виділити такі правила чистої архітектури:

- Незалежність від інтерфейсу користувача. Використовуючи чисту архітектуру, ми повинні мати можливість легко змінювати рівні інтерфейсу користувача або уявлення без зміни рівня програми і т.д. Інтерфейс користувача може бути з будь-якої інтерфейсної платформи або консольного інтерфейсу користувача, будь-якого web-сайту і може бути замінений без зміни інших шарів або решти системи;
- Незалежна від бази даних. Архітектура має бути досить гнучкою, щоб змінювати бази даних, не впливаючи на варіанти використання програм та сутності. Рішення може перемикати набір даних на MS SQL, SQLite, PostgreSQL;

– Незалежність від фреймворку. Основні бізнес-правила або правила програми не повинні залежати від існування фреймворків та бібліотек у майбутньому. Ми можемо включати фреймворки, але як інструменти і рішення не повинно повністю покладатися на них;
– Тестована архітектура повинна відповідати вимогам тестування основної програми та бізнес-кейсів та правил без інтерфейсу користувача, бази даних, web-сервера або будь-якого зовнішнього компонента;

Як правило, чиста архітектура розкривається за допомогою основної кругової ілюстрації, представленої Робертом Мартіном на Рисунку 2.7.

На цій діаграмі чудово показані концепції високого рівня, що лежать в основі чистої архітектури.

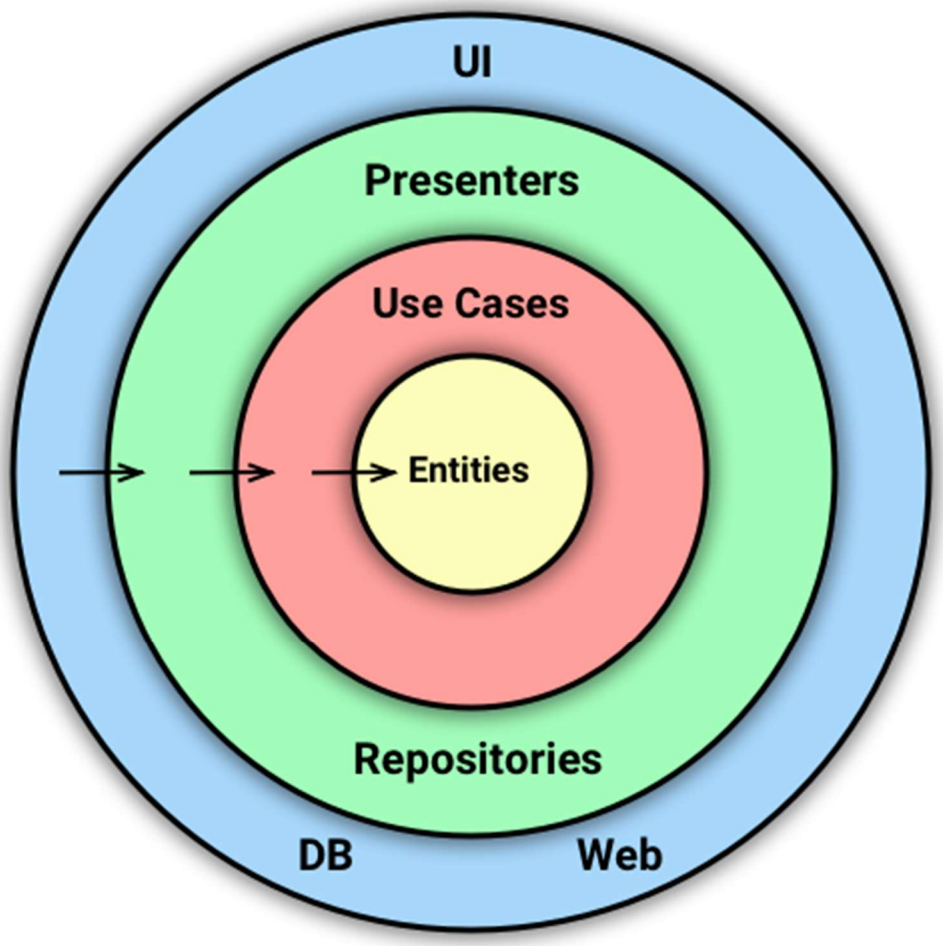


Рисунок 2.7 – Clean Architecture за Робертом Мартіном

З діаграми видно, що сутності є внутрішнім ядром, яке зазвичай називають сутністю бізнес-домену. Його також називають корпоративними бізнес-правилами.

Ці бізнес-об'єкти охоплюються варіантами використання, також званими бізнес-правилами додатків. Ці варіанти використання викликаються з презентаторів, контролерів або шлюзів, як показано на діаграмі вище. Додаткові зовнішні інтерфейси, БД, інтерфейс користувача або мережа, які зазвичай називаються загальнодоступною оболонкою, загальнодоступною поверхнею або інтерфейсом. Всі фреймворки та драйвери знаходяться на зовнішніх рівнях. Крім того, ми можемо спостерігати, що потік йде від загальнодоступної оболонки до внутрішньої сутності.

Якщо розуміти з іншого боку, то залежності переміщуються від внутрішнього до зовнішнього, тобто від ядра до зовнішньої чи публічної поверхні. Основні внутрішні об'єкти та варіанти використання, також звані бізнес-рівнями та рівнями додатків, не мають залежностей та з меншою ймовірністю зміняться. Кожен шар цієї кругової діаграми залежить від шару поруч із ним. Зовнішні рівні, швидше за все, зміняться залежно від технологій, фреймворків тощо, отже, архітектура рішення менше впливає логіку основних додатків.

Слід зазначити, що Clean Architecture була придумана для великих серверних програм для великого бізнесу, а не для додатків, які не потребують подальшого розвитку. Також потрібно розуміти, що шарів не обов'язково буде 4, можливо менше, головне зберігати логіку Clean Architecture.

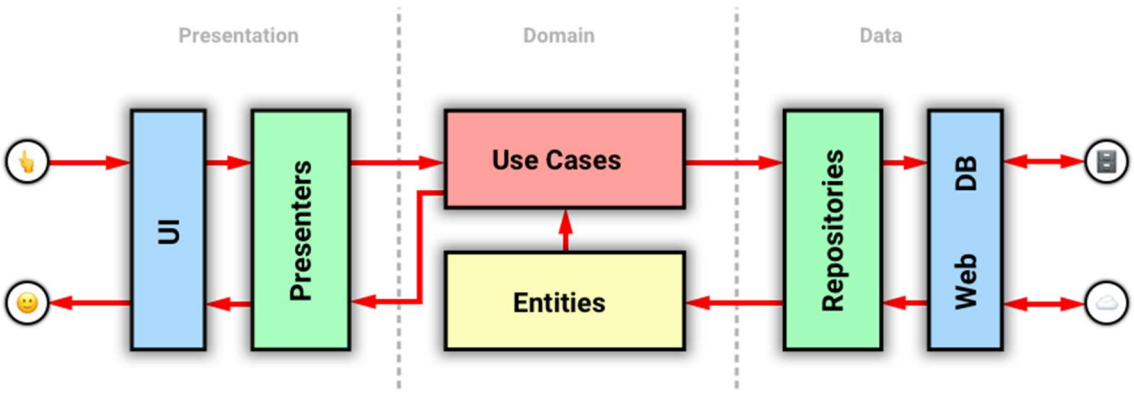


Рисунок 2.7 – Потік даних

Події користувача переходять до Presenter, який передається у Use Cases. Use case робить запит до репозиторію. Репозиторій десь отримує дані, створює Entities, передає їх у Use Cases. Таким чином, Use cases отримує всі необхідні сутності. Потім, застосовуючи їх і свою логіку, він отримує результат і передає його Presenter. У свою чергу, він відображає результати в інтерфейсі користувача.

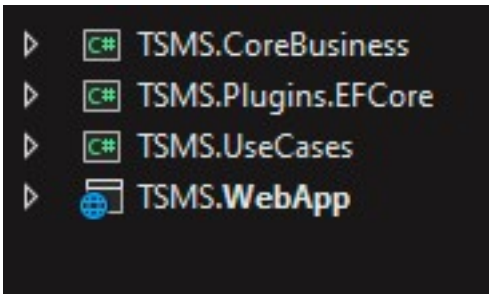


Рисунок 2.8 – Архітектура web-додатку

Так створюючи web-додаток ми можемо виділити 4 шара на Рисунку 2.8.

В нас є 4 шара (CoreBusiness, Plugins.EFCore, UseCases, WebApp) де CoreBusiness це Entities, Plugins.EFCore це Repositories і WebApp це UI.

Підсумовувати все це можна зробити висновки що:

- шар Entities містить об'єкти, які реалізують логіку бізнесу, загальну додатку;
- шар Repositories надає доступ до даних;
- Use-Case – це деталізація, опис дії, яку може зробити користувач системи;
- UI - цей рівень по суті містить компоненти вводу-виводу системи, графічний інтерфейс користувача;

3.ПРОГРАМНА РЕАЛІЗАЦІЯ ДОДАТКУ

3.1. Розробка концепції

Аналізуючи аналоги та актуальність додатку, для найефективнішого розповсюдження було прийняте рішення його розробки як web-додаток, наступним етапом стане проголошення концепції.

Цільова аудиторія – малий та середній бізнес, будь якого направлення, так як цей web-додаток створений для вирішення деяких бізнес процесів.

Головна ідея цього web-додатку полягає в контролі складу та торгівлі, які є найважливішими частинами малого та середнього бізнесу.

Короткий опис застосування – цей web додаток націлений на контроль товарів на складах та їх керування у додатку, та контроль торгівлі. За його допомогу ви зможете керувати товарами/продуктами, можливість редагувати/видаляти/створювати, також розподіл їх за категоріями та брендами, та створення прайс-листів. Також ви маєте можливість керувати торгівлею, тобто створювати закупки та продажі товарів і відслідковувати затрати та доходи для статистики.

Головні вимоги додатку:

- Створення нових позицій товарів та можливість редагування;
- Створення брендів та категорій товарів для розподілу, та подальша можливість редагування;
- Створення закупок;
- Створення продаж;
- Списки продаж та закупок;
- Візуальне інформування по затратам за доходам;

3.2. Оцінка та планування розробки web-додатку

Так як ми визначили набір вимог та архітектуру системи, потрібно було сформувати список задач, для того щоб наш web-додаток обов'язково мав функціонал, який ми визначили раніше, для того щоб отримати Minimum Viable Product. Тому, що для реалізації web-додатку були сильно обмежені часові рамки , а можливості web-додатку таких систем майже безмежні, якщо враховувати можливість інтеграцій, інших систем по API. Тому для ефективної розробки потрібно було створити рамки, які б виконували потрібній основний функціонал для цієї сфери.

Так як, в нас web-додаток з гранично чіткими вимогами та заздалегідь продуманими способами реалізації, то нам потрібно було створити відстежування задач для полегшення контролю по виконаним задач, для чіткого розуміння, що потрібно виконати ще.

На даний момент існує безліч web-додатків для контролю проектами, такі як Trello, Monday.com, Asana, Wrike. Був же вибраний додаток Asana, тому що, ви можете легко спланувати ти та організувати кроки, необхідні вашій команді, щоб виконати ваші терміни. В Asana у вас є можливість створювати дошки із призначеними завданнями та підзавданнями. Є багато варіантів для простого сортування за цими завданнями, у тому числі за їхніми термінами.

Всі вимоги, які були виявлені, були створені в Asana, для відстежування статусу їх виконання та для розуміння створених дедлайнів, для того щоб впоратись в виділений час (Рис. 3.1).

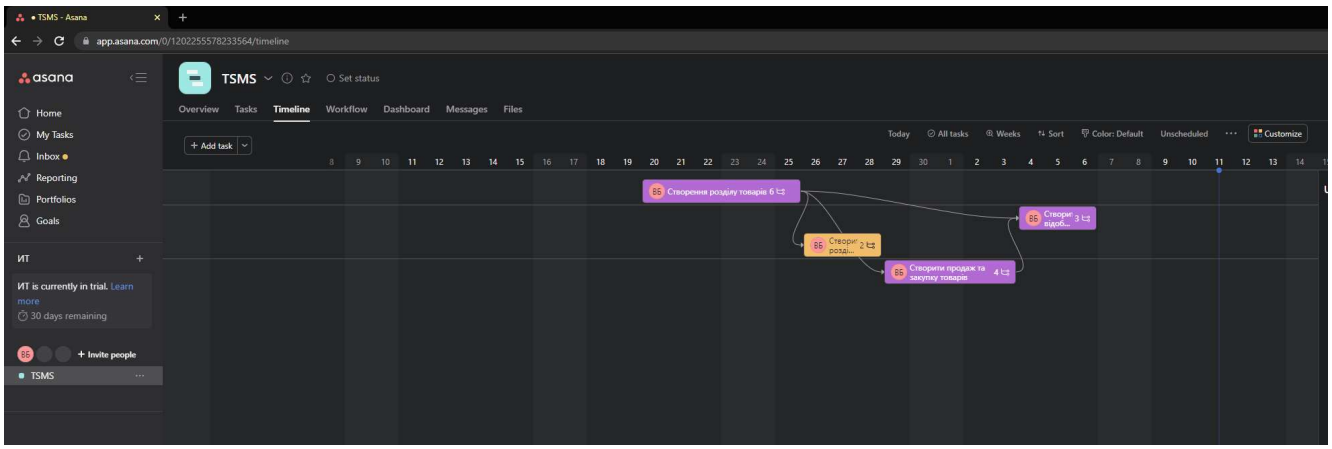


Рисунок 3.1 – Відстеження статусу розробки продукту за допомогою інструмента Asana

Також Asana дає змогу створити залежності конкретної задачі від іншої (Рис 3.1), для того щоб мати змогу при деяких труднощах мати змогу перейти на іншу частину розробки, на яку є можливість, для того щоб не витрачати час.

Кожен вимога містить задачі, які були поміщені в систему, для більш конкретної розробки функціоналу, та надає змогу розбивати конкретну задачу на низку технічних підзадач (Рис 3.2).

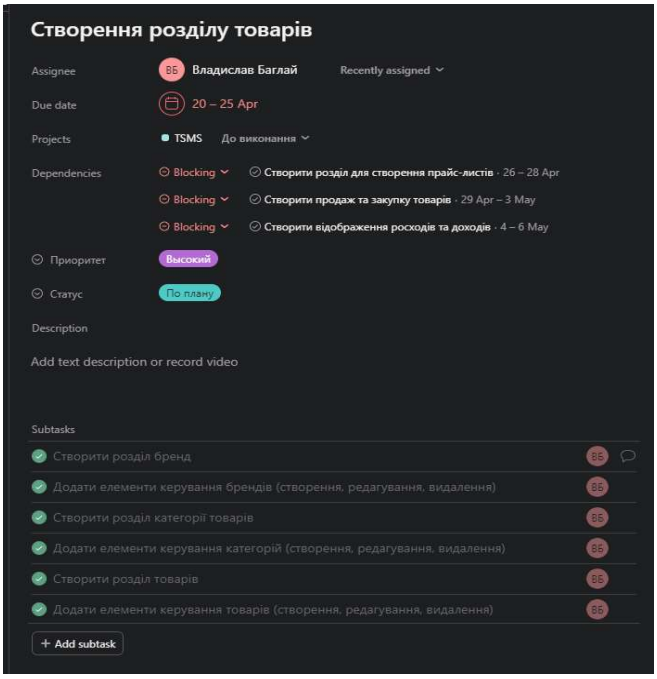


Рисунок 3.2 – Приклад перегляду створеної вимоги розробки

Кожна вимога описана у стилі Use-Case, що дозволяє застосувати підхід BDD (Behaviour Driven Development - "Розробка через поведінку") – це методологія розробки програмного забезпечення, яка базується на розробці на основі функцій, зосереджуючись на сценаріях використання та поведінці системи.

Також до кожної підзадачі можна створити ще одну підзадачу, якщо того потребує розділення вимог.

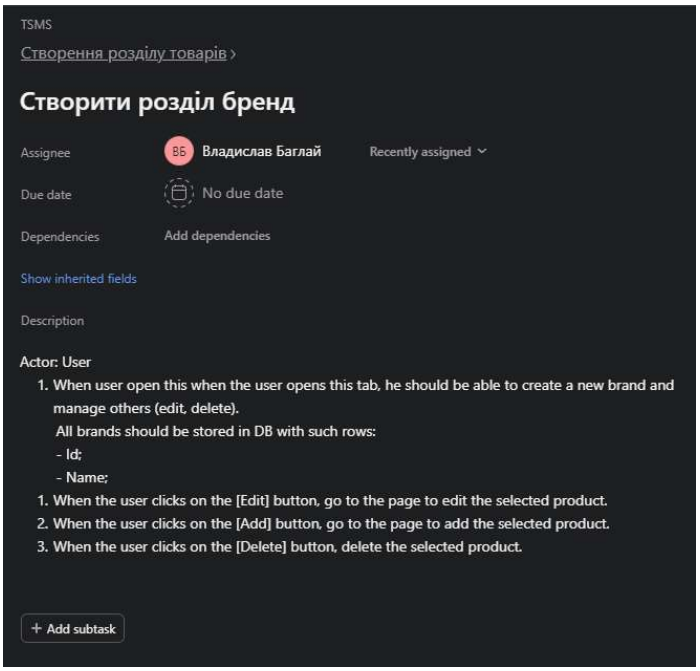


Рис 3.3. – Приклад описання підзадачі web-додатку у форматі Use-Case

Також до кожної підзадачі можна створити ще одну підзадачу, якщо того потребує розділення вимог.

3.3. Набір програмних засобів які використовуються для розробки

В якості IDE було вибрано Visual Studio 2022 оскільки тільки з нею можливо працювати з платформою ASP.NET Core Blazor а також написання коду програм з використанням різноманітних технологій та різних мов. Основні переваги Visual Studio по зрівнянню з іншими IDE це:

– Один із найзручніших інтерфейсів;

– Підтримка багатьох мов програмування та платформ;
– Пряма підтримка Git та GitHub;
– Можливість працювати з базами даними напряму та з SQL запитами;
– Підтримка багатьох плагінів, бібліотек та розширень за допомогою NuGet
– Наявність шаблонів для різноманітних проектів

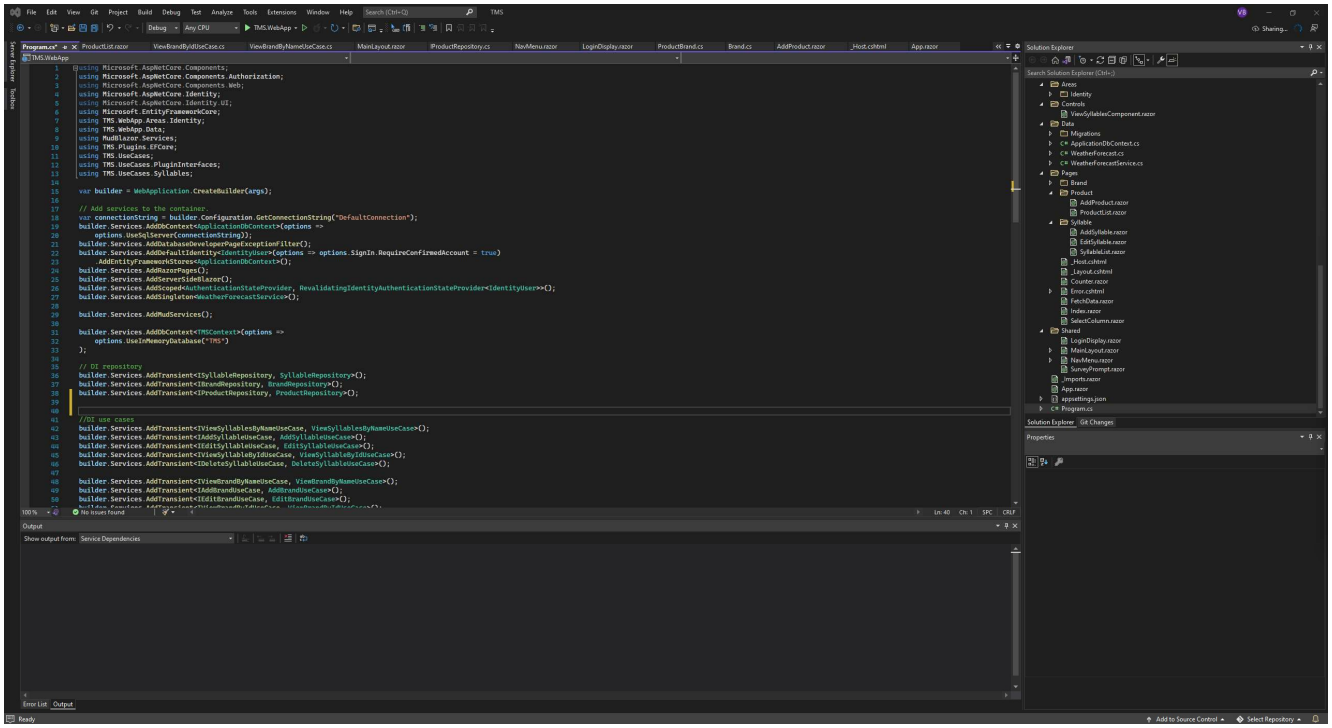


Рис 3.4. – Visual Studio 2022

Так як, ми використовували для побудування бази даних Entity Framework Core, то ми мали вибір, як будувати БД, з трьох методів (Code-First, Model-First, Database-First). Code-First найкращий вибір, так як ми будуємо БД з коду, що набагато швидше ніж два других метода. За допомогою цього метода, ми навіть можемо не переглядати БД в інструментах, але це потрібно для перевірки коректності створення БД, для того щоб не було технічних проблем. Проблема може бути в створених зв'язка або в типу даних, так як типи даних C# конвертуються в SQL Server.

Тому для перегляду та тестування даних в БД, було використано 2 інструмента SQL Server Management Studio і Server Explorer від Visual Studio.

SQL Server Management Studio – це інтегроване середовище для управління будь-якою інфраструктурою SQL. Дана утиліта надає змогу для доступу, налаштування, управління, адміністрування та розробки всіх компонентів SQL Server. За допомогою цього додатку ми можемо створювати та керувати декількома БД. Також є можливість працювати з Azure Data Studio для хмарного зберігання БД (Рис 3.5).

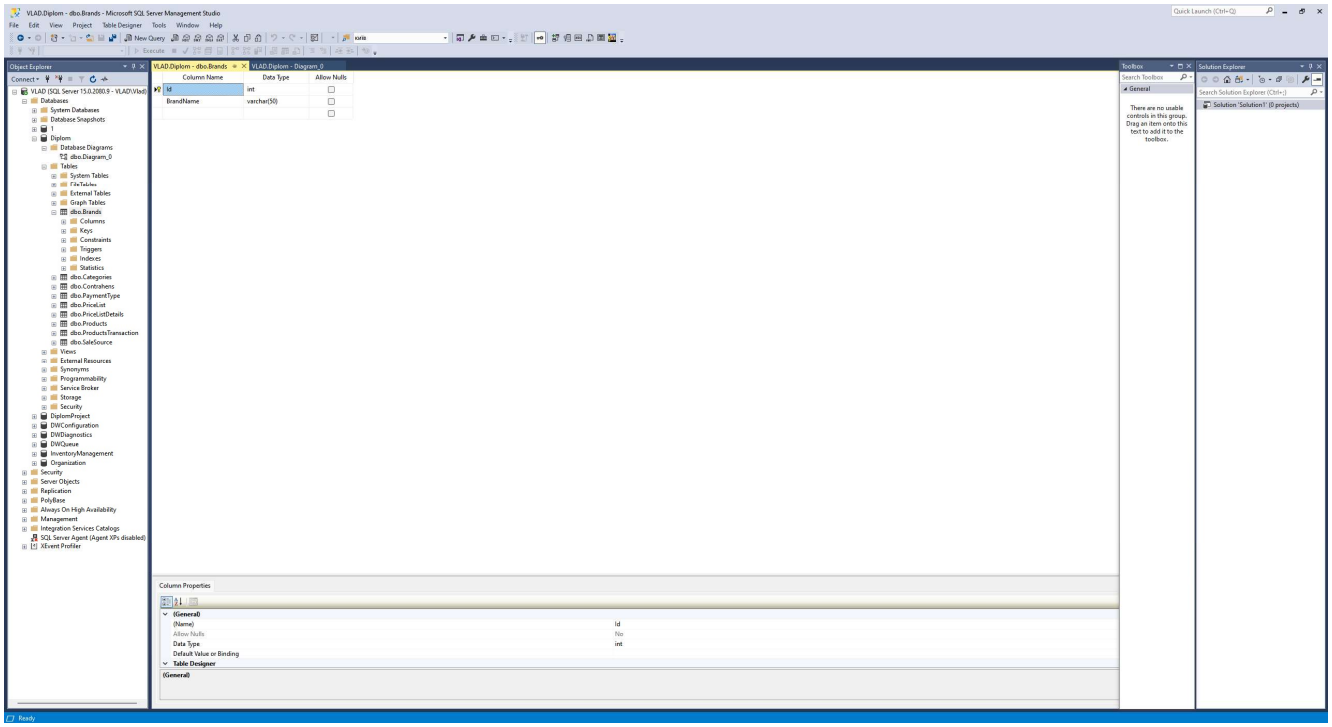


Рисунок 3.5 – Microsoft SQL Server

Server Explorer від Visual Studio – це інструмент, який надає змогу прямо в IDE підключити БД, та досліджувати елементи таблиць та редагувати таблиці при необхідності. Це дуже зручний інструмент для швидкого розгортання БД в IDE для перевірки таблиць в БД. Також можливо копіювати T-SQL запити для створення таблиць, які підтримуються в SQL Server Management Studio (Рис 3.6).

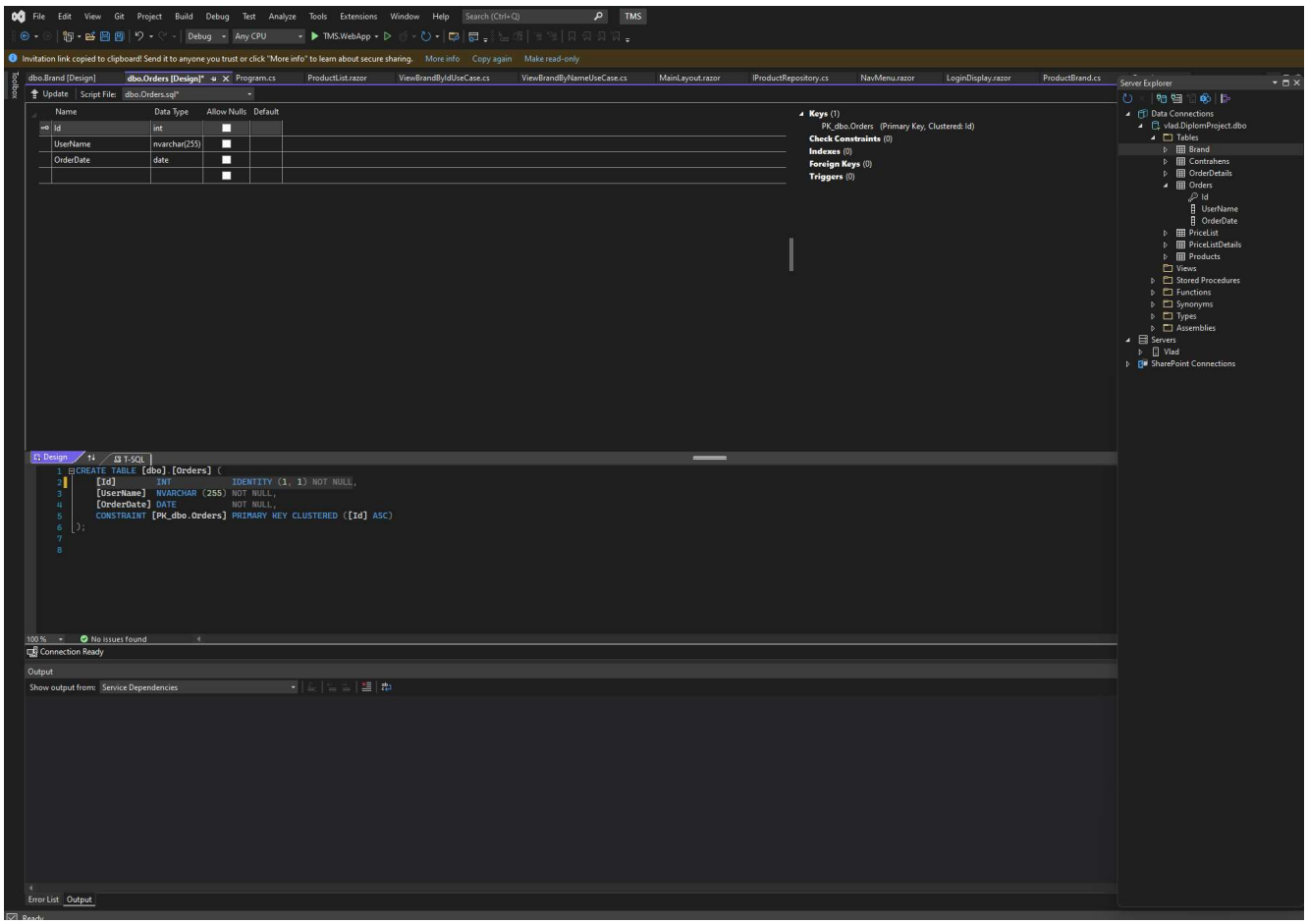


Рисунок 3.6 – Server Explorer

Також потрібно зазначити, що для цих двох інструментів не потрібно обов’язково знати SQL або T-SQL мову запитів, так як в нас є можливість працювати через інтерфейс та створювати БД та таблиці за допомоги діаграм, або створювати окремо.

Для відстежування статусу задач та створення вимог, використано web-додаток Asana. Цей додаток надає змогу документувати весь процес розробки малих проектів. Asana має дуже простий та зручний інтерфейс, а також багато функціоналу для менеджменту проектів. Дає змогу працювати на декількох проектах, створювати портфоліо, також має мобільний додаток. Головний мінус цього додатку є те що він має підписну форму оплати, але додаток коштує тих грошей (Рис 3.7).

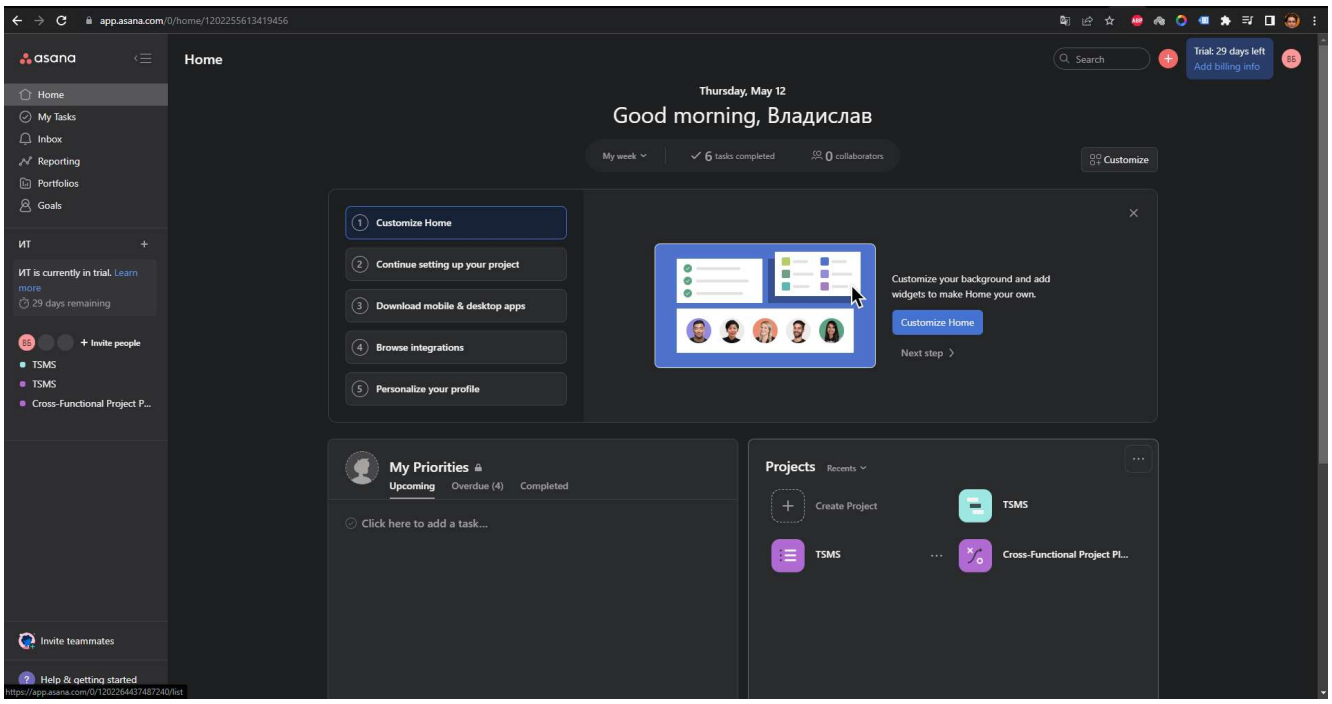


Рисунок 3.7 – Asana

3.4. Функціонал web додатку та його структура

Як вже відомо з раніше створених вимог до системи, web-додаток має такий основний функціонал:

- Створення нових позицій товарів та можливість керування ними;
- Створення брендів та категорій товарів для сортування товарів;
- Створення закупок товарів;
- Створення продаж товарів;
- Створення прайс-листів з можливостю імпорту в .xslx форматі
- Відображення продаж та закупок;
- Візуальне інформування по затратам за доходам;

Так як основна суть web-додатку це робота з даними, для цього щоб зрозуміти нам потрібно розглянути БД, яка була створена за допомогою фреймворка Entity Framework Core (Рис. 3.8).

Рисунок 3.8 – БД web-додатку

Для того щоб зрозуміти, як працює web-додаток, для початку розглянемо дану БД, яка складається з 11 таблиць, більш детальніше:

- Brands. Дана таблиця зберігає та надає інформацію о брендах товарів;
- Categories. Таблиця зберігає та надає о категоріях товарів, потрібна для того щоб фільтрувати товари;
- Products. Дана таблиця це основний елемент web-додатку, так як, , зберігає в собі основну інформацію о товарі (назва, код/артикул, закупна ціна, та бренд і категорія ;
- Purchase. Це також одна із основних таблиць створеного web-додатку, створена для того щоб керувати інформаціє о закупках товарів;
- Order. Таблиця зберігає інформацію о замовленні, та зв’язана з таблицею OrderDetail, яка надає більш конкретну інформацію;
- OrderDetail. Ця таблиця розкриває всю інформацію о замовленні (товари, які куплені, кількість, за яку ціну);

– PaymentType. Дана таблиця зберігає інформацію о доданих типів оплати;
– SaleSource. Таблиця зберігає інформацію о створених джерелах продажу;
– Contrahens. Таблиця створена для прайс-листів, для сортування для якого контрагента (іншої сторони) створений прайс-лист;
– PriceList. Ця таблиця є «посиланням» на PriceListDetails, та надає змогу імпортувати PriceListDetails в .xslx форматі, для надавання інформації контрагентам;
– PriceListDetails. Дана таблиця зберігає інформацію о товарів з визначеною ціною;

Далі, щоб розглянути функціонал web додатку, потрібно розібрати діаграму класів, яка дає нам найповніше і розгорнуте уявлення про зв'язки в програмному коді, функціональності та інформації про окремі класи (Рис. 3.9).

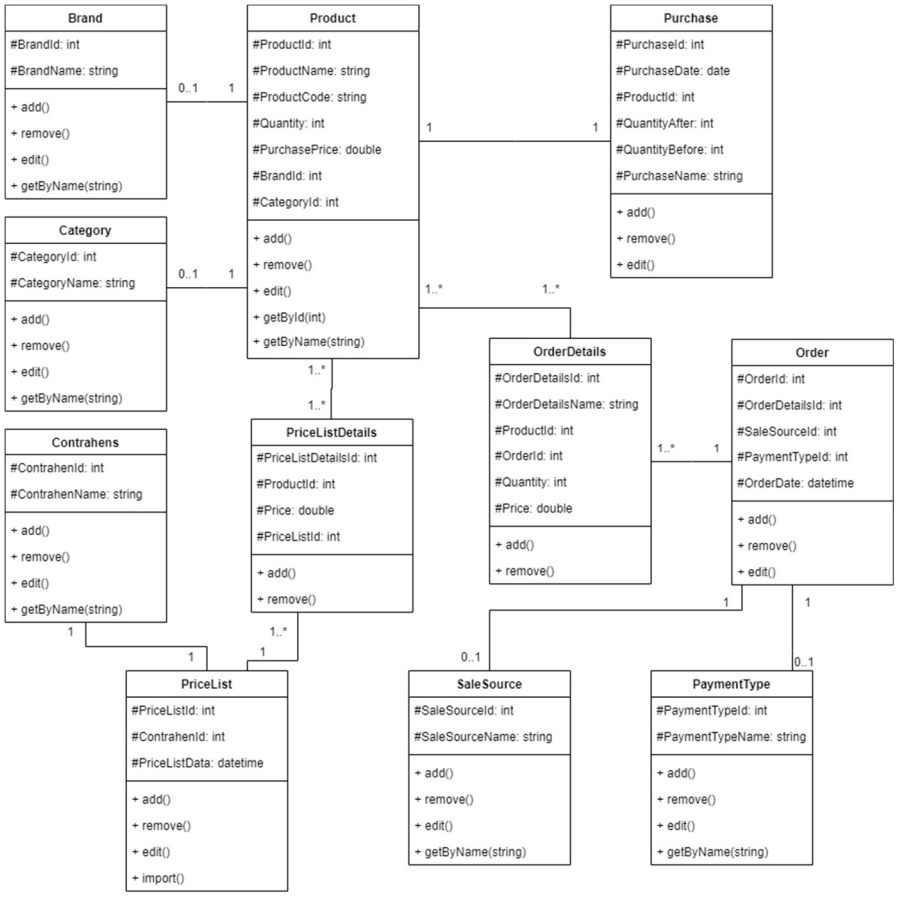


Рисунок 3.9 – Діаграма класів web-додатку

Аналізуючи дану діаграму класів ми отримуємо більше повне уявлення, ніж розглядаючи БД, про взаємозв'язки, методи та видимість членів класів (атрибутів, функцій):

- Так задана видимість для атрибутів – це # (Protected) і для методів – це + (Public);
- Так якщо зрівняти з БД (Рис 3.8), то є можливість розглянути взаємозв'язки між класами, особливо силу відношень;
    a) 0..1. Нуль або один екземпляр;
    b) 1. Обов'язково один екземпляр;
    c) 0..* або *. Нуль або більше екземплярів;
    d) 1..*. Один обов'язково або більше екземплярів;

Також потрібно зазначити особливість додатку. Так як в нас Clean Architecture, і самі класи не зберігають ніякі функції класів, так як самі класи знаходяться в шарі Entity. Самі методи зберігаються в шарі Repositories, так як цей шар надає доступ до даних, а в додатку весь функціонал – це робота з даними, і в шарі Use-Case вже знаходиться опис дій та самі інтерфейси для роботи з шаром UI.

3.5. Тестування web-додатку

Для перевірки належного рівня якості розробленого web-додатку, та високого рівня надійності розробленої системи, було вирішено протестувати web-додаток за схемою: Зовнішнє -> Внутрішнє -> Стійкість -> Взаємодія.

- Зовнішнє – перевірка зовнішнього вигляду та функцій, які доступні лише звичайному користувачеві (GUI, Usability).
- Внутрішнє – всі функції програми (Functional).
- Стійкість - сюди ми віднесемо стійкість додатку до навантажень за допомогою Apache JMeter і спроб порушити його безпеку.
- Взаємодія - робота програми з іншими браузерами.

Тестування GUI. Так як не було створений макет додатку і нам не було з чим зрівняти зовнішній вигляд, то в тестування GUI було включено: контент на наявність орфографічних помилок, поява курсору в текстових полях, стандарти

HTML/CSS та масштабованість. Якщо говорити про перевірку GUI, то треба зазначити, що ми використовували бібліотеку MudBlazor, і це означає, що ми не використовували HTML/CSS для відображення контенту (Рис 3.10).

```
<MudDataGrid T="Inventory" MultiSelection="true" Items="@listInventories" Sortable="true" Filterable="true" QuickFilter="@_quickFilter" ShowColumnOptions = "true" >
    <ToolBarContent>

        <MudText Typo="Typo.h4" Class="mr-5">Products</MudText>
        <MudTooltip Text="Create">
            <MudButton Variant="Variant.Filled" Color="Color.Tertiary" OnClick="@AddInventory">Create</MudButton>
        </MudTooltip>
        <MudSpacer />
        <MudTextField @bind-Value="_searchString" Placeholder="Search" Adornment="Adornment.Start" Immediate="true"
            AdornmentIcon="@Icons.Material.Filled.Search" IconSize="Size.Medium" Class="mt-0"></MudTextField>

    </ToolBarContent>
    <Columns>
        <Column T="Inventory" Field=@nameof(Inventory.InventoryName) Title="Name" />
        <Column T="Inventory" Field=@nameof(Inventory.Quantity) Title="Quantity" />
        <Column T="Inventory" Field=@nameof(Inventory.Price) Title="Price" />
        <Column T="Inventory" CellClass="d-flex justify-end">
        <CellTemplate>
            @{

            }
            <MudPaper Elevation="60">
                <MudTooltip Text="Edit">
                    <MudFab Color="Color.Warning" StartIcon="@Icons.Material.Filled.Edit"  OnClick="@(()=> EditInventory(1))" Size="Size.Small" />
                </MudTooltip>
                <MudTooltip Text="Delete">
                    <MudFab Color="Color.Secondary" StartIcon="@Icons.Material.Filled.Delete" OnClick="@(()=> DeleteInventory(1))"  Size="Size.Small" />
                </MudTooltip>
            </MudPaper>
        </CellTemplate>
        </Column>
    </Columns>
    <PagerContent>
        <MudDataGridPager T="Inventory" />
    </PagerContent>
</MudDataGrid>
```

Рисунок 3.10 – Приклад коду з використанням MudBlazor

Ця бібліотека дозволяла зекономити час на візуальній частині web-додатку. І поява курсору в текстових полях, стандарти HTML/CSS та масштабованість та інші частини GUI (кодировка, кроссбраузерність) немає сенсу для перевірки. Тому з GUI був перевірений лише контент на орфографічні помилки.

Після GUI нам потрібно було перевірити web-додаток на Functional. Так як ми мали створені, діаграму станів та діаграму прецедентів, на їх основі було вирішено створити тест-сети для перевірки функціональності нашої програми.

Test Name:	Log in to the system		
Status:	Succeeded		
Created Date:	10/5/2022		
Designer:	Vladyslav		
Pre conditions: Login vladyslav@gmail.com Password: Cdas13ndas			
Steps	Description	Expected Result	Status
Step 1	Navigate to web-application	The login and registration page will open.	Pass
Step 2	Enter your login in the login field	Will not throw an error	Pass
Step 3	Enter your password in the password field	Password will not be visible	Pass
Step 4	Click on submit button	the data will be accepted and will be navigate to the home pa	Pass
Test Name:	Create brand		
Status:	Succeeded		
Created Date:	10/5/2022		
Designer:	Vladyslav		
Pre conditions: authorized user on home page			
Steps	Description	Expected Result	Status
Step 1	Click on menu	tabbed list will appear	Pass
Step 2	Click on products	the list expands to : brands, category of product, products	Pass
Step 3	Click on brands	Navigate to brands page	Pass
Step 4	Click on create button	Navigate to create brands page	Pass
Step 5	Enter any symbols in text field	The text field must accept all characters entered	Pass
Step 6	Click on add button	Navigate to brands page and appear new brand with name which was entered in text field	Pass

Рисунок 3.11 – Приклад текст-кейсів для тестування системи

Далі нам потрібно провести тест Performance. Було використано додаток Apache JMeter, який надає можливість зробити навантаження на додаток.

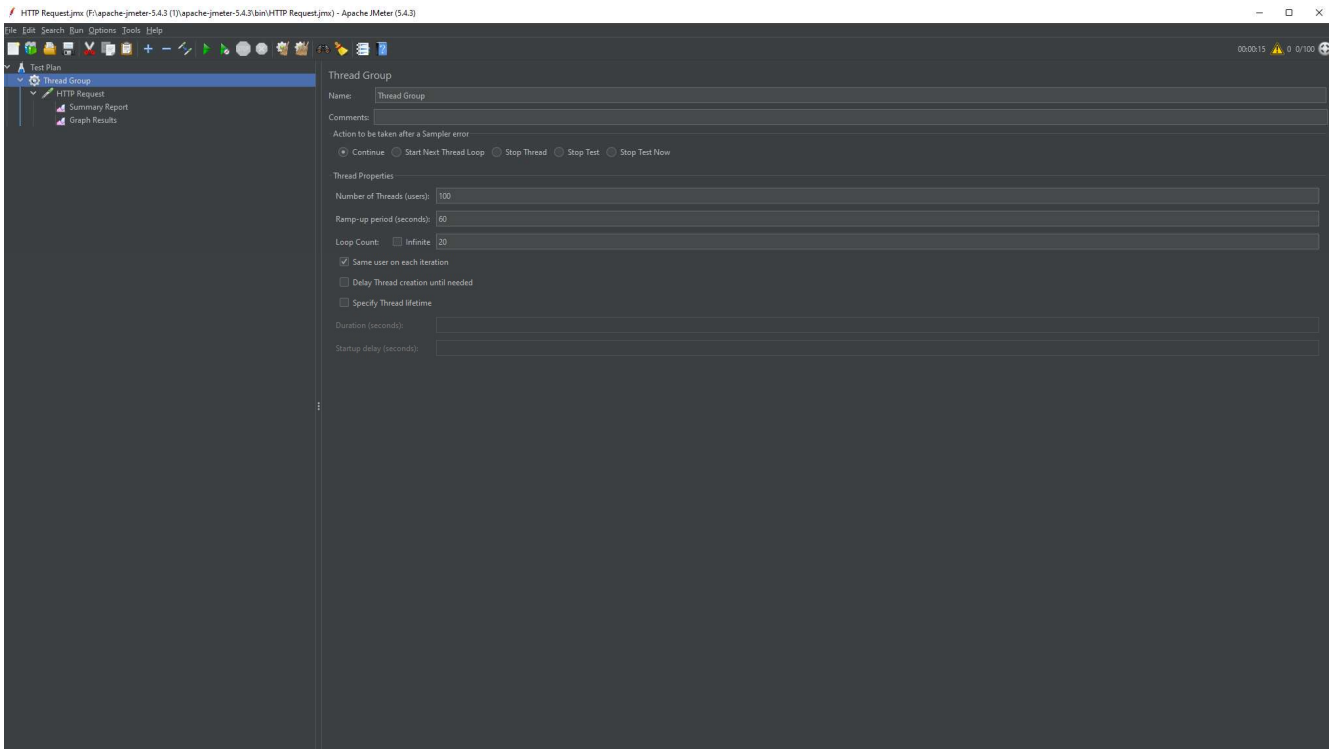


Рисунок 3.12 – Apache JMeter

Залишилось лише провести конфігураційне тестування. Тут все дуже просто, так як в нас web-додаток, нам потрібно мати список з якими браузерами буде працювати розроблений додаток. Ми використовували Blazor, який працює за допомогою Web Assembly.

Рисунок 3.13 – Список браузерів які підтримують Blazor

Також на етапі розробці були проведені статичні тестування (статичний аналіз), для того щоб виявити:

– змінні, які не використовуються;
– мертвий код;
– нескінченні цикли;
– змінні з невизначеними значеннями;
– неправильний синтаксис;

ВИСНОВКИ

У результаті виконання даної дипломної роботи було спроектовано та реалізовано web-додаток, за допомогою якого можна підвищити ефективність керування торгівлею та складським обліком.

1. Проведено аналіз існуючих додатків для визначення переваг та недоліків існуючих систем. Було вирішено розробити web-додаток, через його розповсюдження, найкращу відповідність сучасним вимогам й найбільшим потенціалом успіху в майбутньому. Для покращення розуміння можливостей та функцій, які реалізуватиме web-додаток, було проведено аналіз існуючих систем для керування торгівлею та складським обліком.

2. Проведено аналіз інструментів та програмних засобів реалізації для реалізації web-додатка, які впораються зі створенням функціоналу додатку.

Було вибрано інструменти для побудови web-додатку, такі як ASP.NET Core Blazor, Entity Framework Core, SQL Server, Microsoft Visual Studio 2022 як середовища розробки, обґрунтовано їх перевагу та ефективність над іншими аналогами

Для того щоб розробити успішний продукт, було проаналізовано аналоги та створено набір вимог у вигляді UML діаграм

3. Описано програмні засоби та інструменти, котрі було застосовано для розробки програмного забезпечення. Виявлено що середовище розробки Visual Studio 2022 є найбільш зручним для виконання поставлених задач. Підібрано допоміжні інструменти розробки, такі як SQL Server для тестування БД. Менеджмент та відстеження статусу виконання проекту було здійснено за допомогою інструменту Asana.

4. Зроблена система цілеспрямована на допомогу малому бізнесу, в сфері роздрібної торгівлі. Було з’ясовано що найкраще інструмент підходить для вирішення таких стандартних бізнес-процесів, як склад та продажі. Визначено, що web-додаток може використовуватися як і для початківців, які тільки розпочали власний бізнес, та вже для компетентних підприємців.

Створений web-додаток має значні перспективи подальших досліджень. За допомогою ASP.NET Core Blazor можна легко розширювати функціонал додатку, по мірі росту та потреб користувачів.

ПЕРЕЛІК ПОСИЛАНЬ

1. Andrew Troelsen, Phil Japikse, Pro C# 7: With .NET and .NET Core /- Apress – с. 403 – 420.
2. Opendatabot [Електронний ресурс] : [Web-сайт] – Режим доступу: https://opendatabot.ua
3. Мінфін [Електронний ресурс] : [Web-сайт] – Режим доступу https://minfin.com.ua
4. Приватний підприємець [Електронний ресурс] : [Web-сайт] – Режим доступу http://chp.com.ua
5. Metanit [Електронний ресурс] : [Web-сайт] – Режим доступу: https://metanit.com/
6. Microsoft Docs [Електронний ресурс] : [Web-сайт] – Режим доступу https://docs.microsoft.com/ru-RU/aspnet/core/?view=aspnetcore-6.0
7. John Skeet. C# for professionals: subtleties of programming, 3rd edition, new translation = C# in Depth, 3rd ed.. - M .: "Williams", 2014. - 608 p. - ISBN 978-5-8459-1909-0.
8. Christian Nagel et al. C# 5.0 and .NET 4.5 for Professionals = Professional C# 5.0 and .NET 4.5. - M .: "Dialectics", 2013. - 1440 p. - ISBN 978-5-8459-1850-5.
9. Joshi Bipin. "Blazor." Beginning Database Programming Using ASP. NET Core 3. Apress, Berkeley, CA, 2019. 337-380.
10. Price Mark J. C# 9 and. NET 5–Modern Cross-Platform Development: Build intelligent apps, websites, and services with Blazor, ASP. NET Core, and Entity Framework Core using Visual Studio Code. Packt Publishing Ltd, 2020.
11. Freeman Adam. Pro ASP. NET Core 3: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. Apress, 2020.
12. Aponte Michele. "Blazor Server vs. Blazor WebAssembly." Building Single Page Applications in. NET Core 3. Apress, Berkeley, CA, 2020. 19-31.
13. Himschoot Peter. Blazor Revealed. Apress, 2019.
14. Vermeir Nico. "Blazor." Introducing. NET 6. Apress, Berkeley, CA, 2022. 125-152.

15. Haas, Andreas, et al. "Bringing the web up to speed with WebAssembly." Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2017.
16. Jangda, Abhinav, et al. "Not So Fast: Analyzing the Performance of {WebAssembly} vs. Native Code." 2019 USENIX Annual Technical Conference (USENIX ATC 19). 2019.
17. Lehmann, Daniel, Johannes Kinder, and Michael Pradel. "Everything Old is New Again: Binary Security of {WebAssembly}." 29th USENIX Security Symposium (USENIX Security 20). 2020.
18. Watt, Conrad, Andreas Rossberg, and Jean Pichon-Pharabod. "Weakening webassembly." Proceedings of the ACM on Programming Languages 3.OOPSLA (2019): 1-28.
19. Rossberg, Andreas, et al. "Bringing the web up to speed with webassembly." Communications of the ACM 61.12 (2018): 107-115.
20. Yan, Yutian, et al. "Understanding the performance of webassembly applications." Proceedings of the 21st ACM Internet Measurement Conference. 2021.
21. Martin, Robert C., et al. Clean architecture: a craftsman's guide to software structure and design. No. s 31. Prentice Hall, 2018.
22. Brown, Paul S., et al. "Refactoring the Whitby Intelligent Tutoring System for Clean Architecture." Theory and Practice of Logic Programming 21.6 (2021): 818-834.
23. Joshi, Bipin. "ASP. NET Core Razor Pages." Beginning Database Programming Using ASP. NET Core 3. Apress, Berkeley, CA, 2019. 133-174.
24. Freeman, Adam. "Using Razor Pages." Pro ASP. NET Core 3. Apress, Berkeley, CA, 2020. 557-584.
25. Freeman, Adam. Pro ASP. NET Core 3: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. Apress, 2020.
26. Core Razor Pages, Angular, and Anthony Giretti. "Beginning gRPC with ASP .NET Core 6."
27. Mishra, Atul. "Critical comparison of PHP and ASP .NET for web development." International Journal of Scientific & Technology Research 3.7 (2014): 331-333.

28. Smith, Jon. Entity Framework core in action. Simon and Schuster, 2021.
29. Schwichtenberg, Holger. "Modern Data Access with Entity Framework Core." Modern Data Access with Entity Framework Core. Apress (2018).
30. Schwichtenberg, Holger. "Introducing entity framework core." Modern Data Access With Entity Framework Core. Apress, Berkeley, CA, 2018. 1-14.

ДОДАТКИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Розробка web-додатку для керування торгівлею та складського обліку на платформі ASP.NET Core Blazor

Виконав студент 4 курсу
групи ПД-42
Баглай Владислав Олегович
Керівник роботи
Негоденко Олена Василівна

Київ – 2022

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи –** підвищити ефективність складського обліку поставки та реалізації товарів.
- **Об'єкт дослідження –** процес керування торгівлею та складським обліком.
- **Предмет дослідження –** технології розробки web-додатків за допомогою платформи ASP.NET Core Blazor.

АКТУАЛЬНІСТЬ РОБОТИ

- На сьогоднішній день багато людей вирішують створити свій власний бізнес. З початком створення власного бізнесу підприємці-початківці стикаються з проблемами контролю основним бізнес-процесів. Тому з'являється попит на такі системи для контролю бізнес-процесів. Так як автоматизація бізнесу існує і її потрібно використовувати

- Так додатки допомагають уникнути такі проблеми підприємств, як затримка, простої, недостачі, що суттєво впливає на підприємство. А також дозволяє зекономити значну частину часу для робітників на рутинних операціях і сконцентруватися на більш важливих задачах.

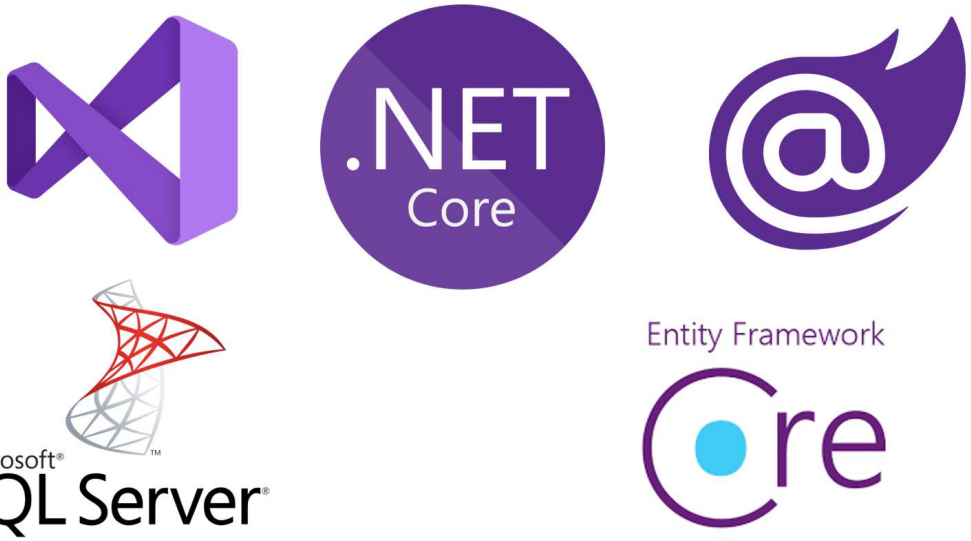
АНАЛОГИ

Назва продукту	Переваги	Недоліки
Дебет Плюс	1) Є безкоштовна версія, з обрізаним функціоналом; 2) Є CRM модуль для роботи з клієнтами;	1) Десктопний додаток. Тому неможливо працювати з інших пристроїв або віддалено; 2) Застарілий інтерфейс з «лишніми кроками» для дій (створення позицій, продаж і т.д);
ERP FOSS	1) Ціна – єдиний тариф, який залежить від кількості користувачів системи. 2) Швидке налаштування для початку роботи. 3) Інтеграція з «Prom.ua» та «Нова пошта».	1) Немає дашбордів, для аналізу продажів. 2) Не зручний інтерфейс, новачок в цій системі може загубитись.
PERFECTUM	1) Має інтеграцію з з IP-телефонією «Binotel». 2) Одноразова оплата. 3) Можливість за додаткову плату модифікувати додаток, що надає безліч можливостей.	1) Ціна. Додаток дуже дорогий, і підходить більш для середнього бізнесу. 2) Непотрібний функціонал. Так як система універсальна, то містить дуже багато непотрібного функціоналу для користувача, який в основному заважає.

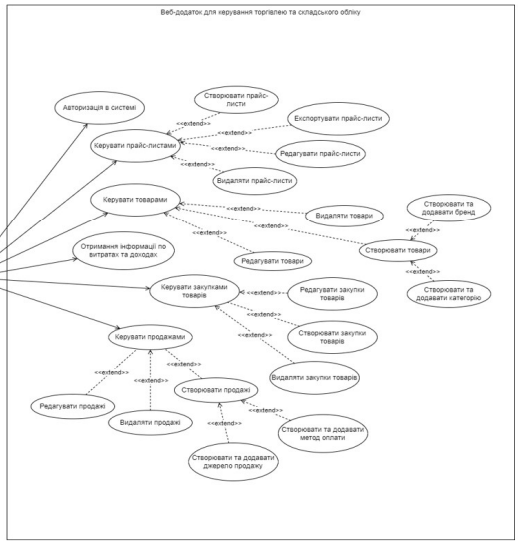
ТЕХНІЧНІ ЗАВДАННЯ

- *Створення нових позицій товарів та можливість редагування;*
- *Створення брендів та категорій товарів для розподілу, та подальша можливість редагування;*
- *Створення закупок;*
- *Створення продаж;*
- *Відображення продаж та закупок;*
- *Візуальне інформування по затратам та доходам;*
- *Створення прайс-листів;*

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



ДІАГРАМА ПРЕЦЕДЕНТІВ СИСТЕМИ



ДІАГРАМА ДІЯЛЬНОСТІ

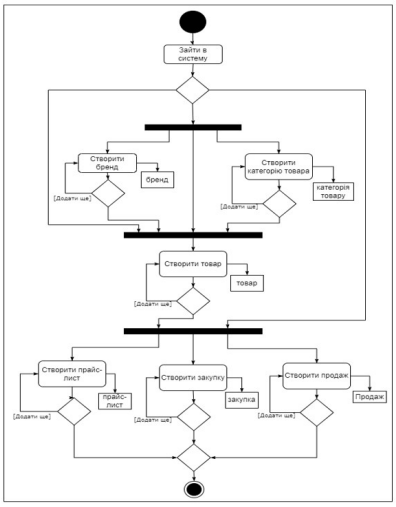
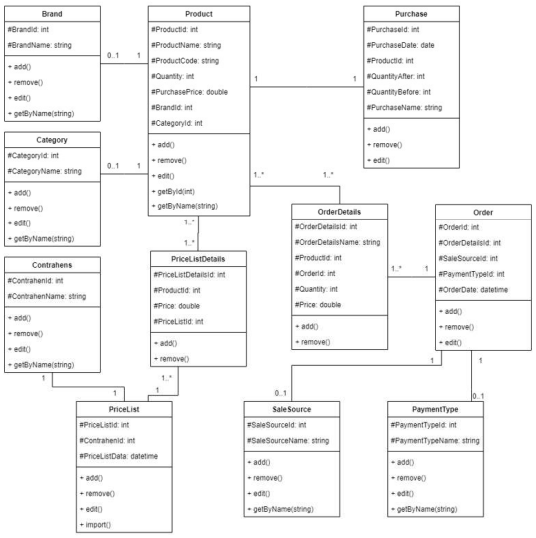


СХЕМА БАЗИ ДАНИХ ВЕБ-ДОДАТКУ

ДІАГРАМА КЛАСІВ



АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

ВИСНОВКИ

1. Проведено аналіз існуючих додатків для визначення переваг та недоліків існуючих систем. Було вирішено розробити web-додаток, через його розповсюдження, найкращу відповідність сучасним вимогам й найбільшим потенціалом успіху в майбутньому.
2. Проведено аналіз інструментів та програмних засобів реалізації для реалізації web-додатка, які впораються зі створенням функціоналу додатку.
3. Описано програмні засоби та інструменти, котрі було застосовано для розробки програмного забезпечення.
4. Зроблена система цілеспрямована на допомогу малому бізнесу, в сфері роздрібної торгівлі. Було з'ясовано що найкраще інструмент підходить для вирішення таких стандартних бізнес-процесів, як склад та продажі.

Перспективи подальших досліджень:

- Можливість інтегрування з іншими системами за допомогою API;
- Розширити можливості контролю створених бізнес процесів та додати нові;
- Можливість завантажувати файли;
- Створення CRM модуля;

ДЯКУЮ ЗА УВАГУ!