Державний торговельно-економічний університет
Кафедра інженерії програмного забезпечення та кібербезпеки

ВИПУСКНИЙ КВАЛІФІКАЦІЙНИЙ ПРОЄКТ

на тему:

«Програмна компонента POS-рішення «Skyservise» для підприємства сільськогосподарського господарства»

Студента 4 курсу, 6 групи,
спеціальності 121 «Інженерія програмного забезпечення»
освітньої програми «Інженерія програмного забезпечення» _____ Максимчук Роман Васильович
підпис студента

Науковий керівник
доктор технічних наук,
професор кафедри інженерії програмного забезпечення та кібербезпеки _____ Хорольська Карина Вікторівна
підпис керівника

Гарант освітньої програми
кандидат технічних наук,
доцент кафедри інженерії програмного забезпечення та кібербезпеки _____ Рзаєва Світлана Леонідівна
підпис гаранта

КИЇВ – 2023

Державний торговельно-економічний університет

Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення та кібербезпеки
Освітній ступінь бакалавр
Спеціальність 121 «Інженерія програмного забезпечення»

Затверджую
Зав. кафедри інженерії програмного забезпечення та кібербезпеки
Криворучко О. В.
«14» листопада 2022 р.

Завдання
на випускний кваліфікаційний проєкт студентові

Максимчук Роман Васильович
(прізвище, ім'я, по батькові)

1. Тема випускного кваліфікаційного проєкту «Програмна компонента POS-рішення «Skyservise» для підприємства сільськогосподарського господарства»

Затверджена наказом ректора від «6» грудня 2022 р. № 3288

2. Строк здачі студентом закінченого проєкту 5 червня 2023

3. Цільова установка та вихідні дані до проєкту

Мета проєкту - розробка програмної компоненти POS-рішення під назвою "SkyService" для підприємств сільського господарства

Об'єкт дослідження - процес автоматизації та управління продажами та складськими запасами на підприємствах сільського господарства.

Предмет дослідження розробка - програмна компонента POS-рішення "SkyService", яка розробляється для оптимізації бізнес-процесів в сільському господарстві, зокрема, управління продажами та складським обліком.

4. Консультанти проєкту із зазначенням розділів, які консультують:

Розділ	Консультант (прізвище, ініціали)	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Зміст випускного кваліфікаційного проєкту (перелік питань за кожним розділом)

ВСТУП

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАСТОСУВАННЯ POS-РІШЕНЬ НА СІЛЬСЬКОГОСПОДАРСЬКИХ ПІДПРИЄМСТВАХ

1.1. Характеристика POS-рішень

1.2. Розробка та впровадження програмної компоненти POS-рішення в сільськогосподарське підприємство

1.3. Технічне завдання

1.4. Висновок до розділу 1

РОЗДІЛ 2. ВИБІР ПРОГРАМНИХ ЗАСОБІВ ТА ПРОЄКТУВАННЯ ПРОГРАМНОЇ КОМПОНЕНТИ

2.1. Чинники, що визначають хід і результати роботи програмної компоненти POS-рішення

2.2 Інструменти розробки програмної компоненти POS-рішення

2.3. Висновок до розділу 2

РОЗДІЛ 3. РОЗРОБКА ТА ВПРОВАДЖЕННЯ ПРОГРАМНОЇ КОМПОНЕНТИ POS-РІШЕННЯ «SKYSERVICE»

3.1. Функціональні характеристики розробленої програмної компоненти

3.2. Особливості впровадження розробленої програмної компоненти

3.3. Висновок до розділу 3

ВИСНОВОК ТА ПРОПОЗИЦІЇ

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

ДОДАТКИ

6. Календарний план виконання проєкту

№ пор.	Назва етапів випускного кваліфікаційного проєкту	Строк виконання етапів проєкту	
		за планом	фактично
1	2	3	4
1.	*Вибір теми випускного кваліфікаційного проєкту*	*21.09.2022*	*21.09.2022*
2.	*Розробка та затвердження завдання на проєкт*	*14.11.2022*	*14.11.2022*
3.	*Вступ та перелік літературних джерел*	*23.12.2022*	*23.12.2022*
4.	*Розділ 1. Аналіз предметної області застосування Pos-рішень на сільськогосподарських підприємствах*	*27.01.2023*	*27.01.2023*
5.	*Розділ 2. Вибір програмних засобів та проєктування програмної компоненти*	*03.03.2023*	*03.03.2023*
6.	*Розділ 3. Розробка та впровадження програмної компоненти Pos-рішення «Skyservice»*	*14.04.2023*	*14.04.2023*
7.	*Висновок*	*28.04.2023*	*28.04.2023*
8.	*Здача випускного кваліфікаційного проєкту на кафедру (перша перевірка)*	*17.05.2023*	*17.05.2023*
9.	*Підготовка автореферату та презентації доповіді*	*26.05.2023*	*26.05.2023*
10.	*Попередній захист випускного кваліфікаційного проєкту*	*29.05.2023 – 02.06.2023*	
11.	*Зовнішнє рецензування випускного кваліфікаційного проєкту*	*05.06.2023*	*05.06.2023*
12.	*Здача прошитого випускного кваліфікаційного проєкту на кафедру*	*05.06.2023*	*05.06.2023*
13.	*Публічний захист випускного кваліфікаційного проєкту*		

7. Дата видачі завдання «14» листопада 2022 р.

8. Науковий керівник випускного кваліфікаційного проєкту Хорольська К.В.
(прізвище, ініціали, підпис)

9. Гарант освітньої програми Рзаєва С.Л.
(прізвище, ініціали, підпис)

10. Завдання прийняв до виконання студент Максимчук Р.В.
(прізвище, ініціали, підпис)

11. Відгук керівника випускного кваліфікаційного проєкту

Науковий керівник випускного кваліфікаційного проєкту

_____

(підпис, дата)

Відмітка про попередній захист _____

(ПІБ, підпис, дата)

12. Висновок про випускний кваліфікаційний проєкт

Випускний кваліфікаційний проєкт студента _____Максимчука Р.В.___

(прізвище, ініціали)

може бути допущена до захисту екзаменаційній комісії.

Гарант освітньої програми _____Рзаєва С.Л._____

(прізвище, ініціали, підпис)

Завідувач кафедри _____Криворучко О. В.

(підпис, прізвище, ініціали)

«_____»_____20 ___ р.

АНОТАЦІЯ

Дослідження присвячене розробці програмної компоненти POS-рішення «Skyservise» для сільськогосподарського підприємства. Шляхом порівняльного аналізу аналогічних рішень були визначені основні вимоги та функціональні можливості системи. Розробка серверної частини була здійснена відповідно до визначених вимог у відповідному середовищі. Готовий програмний комплекс POS-рішення «Skyservise» був успішно протестований, демонструючи свою ефективність та придатність для використання в сільськогосподарських господарствах.

Ключові слова: POS-рішення, Skyservise, сільськогосподарське підприємство, програмна компонента, розробка, тестування.

ABSTRACT

The study is devoted to the development of the software component of the POS solution "Skyservise" for an agricultural enterprise. By means of a comparative analysis of similar solutions, the main requirements and functional capabilities of the system were determined. The development of the server part was carried out according to the defined requirements in the corresponding environment. The finished software complex of POS solution "Skyservise" was successfully tested, demonstrating its effectiveness and suitability for use in agricultural farms.

Keywords: POS solutions, Skyservise, agricultural enterprise, software component, development, testing.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ – програмне забезпечення

XML (англ. Extensible Markup Language) – розширювана мова розмітки

…

					ДТЕУ 121 06-17.БР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		14.04.23	*Програмна компонента POS-рішення «Skyservise» для підприємства сільськогосподарського господарства*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Хорольська К.В.		14.04.23		*ПС*	*7*	*80*
Гарант		Рзаєва С.Л.		14.04.23		*Факультет інформаційних технологій 4 курс, 6 група*		
Розробив		Максимчук Р.В.		14.04.23	*Перелік умовних скорочень*			

ЗМІСТ

ВСТУП

Актуальність. У сучасному сільському господарстві виникає потреба в ефективному управлінні та автоматизації бізнес-процесів. Одним з ключових аспектів є точний облік та контроль за продажами та складськими запасами. В цьому контексті POS-рішення, спеціалізоване програмне забезпечення для точок продажу, стає важливим інструментом для підприємств сільського господарства.

Метою даної дипломної роботи є розробка програмної компоненти POS-рішення під назвою "SkyService" для підприємств сільського господарства. Розглядаються основні функціональні вимоги та особливості, необхідні для ефективного управління продажами, складським обліком та аналізом даних в цій сфері.

Об'єктом дослідження є процес автоматизації та управління продажами та складськими запасами на підприємствах сільського господарства.

Предметом дослідження є програмна компонента POS-рішення "SkyService", яка розробляється для оптимізації бізнес-процесів в сільському господарстві, зокрема, управління продажами та складським обліком.

Для досягнення поставленої мети роботи визначено наступні *завдання*:

- навести характеристику POS-рішень;
- визначити особливості POS-рішень в сільськогосподарській сфері;
- скласти технічне завдання;
- визначити чинники, що визначають хід і результати роботи програмної компоненти POS-рішення;

- дослідити можливі інструменти розробки програмної компоненти POS-рішення;
- визначити функціональні характеристики розробленої програмної компоненти;
- охарактеризувати особливості впровадження розробленої програмної компоненти.

У процесі виконання дослідження будуть використовуватись *методи* аналізу літератури, експертні оцінки, розробка програмного забезпечення, тестування та порівняльний аналіз.

Наукова новизна дослідження полягає у розробці спеціалізованої програмної компоненти POS-рішення "SkyService" для підприємств сільського господарства, що враховує особливості цієї галузі та сприяють покращенню ефективності бізнес-процесів.

Результати цієї роботи будуть мати *практичне значення* для підприємств сільського господарства, які прагнуть автоматизувати та оптимізувати свої бізнес-процеси. Розроблена програмна компонента "SkyService" дозволить ефективніше управляти продажами та складським обліком, забезпечуючи точність та швидкість обробки даних, а також забезпечуючи зручний інтерфейс для користувачів.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ЗАСТОСУВАННЯ POS-РІШЕНЬ НА СІЛЬКОГОСПОДАРСЬКИХ ПІДПРИЄМСТВАХ

1.1. Характеристика POS-рішень

Точка продажу (POS) або точка купівлі (POP) — це час і місце, де завершується роздрібна транзакція. У торговій точці продавець розраховує суму заборгованості клієнта, вказує цю суму, може підготувати рахунок-фактуру для клієнта (який може бути роздруківкою з касового апарату) і вказує варіанти здійснення клієнтом платежу. Це також момент, коли клієнт здійснює платіж продавцеві в обмін на товар або після надання послуги. Отримавши платіж, продавець може видати квитанцію про транзакцію, яка зазвичай роздруковується, але від неї також можна відмовитися або надіслати в електронному вигляді [4; 12].

Щоб обчислити суму боргу клієнта, продавець може використовувати різні пристрої, такі як ваги, сканери штрих-кодів і касові апарати (або більш просунуті «касові апарати POS», які іноді також називають «системами POS»). Для здійснення платежу доступні платіжні термінали, сенсорні екрани та інші апаратні та програмні засоби [5; 11].

Пункт продажу часто називають пунктом обслуговування, оскільки це не лише пункт продажу, але й пункт повернення або замовлення клієнта.

Програмне забезпечення POS-термінала також може містити функції для додаткових функцій, таких як керування запасами, CRM, фінанси або складування.

Підприємства все більше використовують POS-системи, і одна з найбільш очевидних і вагомих причин полягає в тому, що POS-система усуває потребу в цінниках. Ціни продажу пов'язуються з кодом товару під час додавання запасу, тому касир просто сканує цей код, щоб обробити продаж. Якщо ціна зміниться, це також можна легко зробити через вікно інвентарю. Серед інших переваг – можливість реалізації різноманітних видів знижок, схема лояльності для клієнтів, ефективніший контроль за складом. Ці особливості характерні практично для всіх сучасних систем ePOS.

Роздрібні торговці та маркетологи часто називають територію навколо каси місцем покупки (POP), коли вони обговорюють це з точки зору клієнта. Особливо це стосується планування та проектування території, а також розгляду маркетингової стратегії та пропозицій.

Деякі постачальники торгових точок називають свою POS-систему «системою управління роздрібною торгівлею», що є більш відповідним терміном, оскільки це програмне забезпечення призначене не лише для обробки продажів, а й має багато інших можливостей, таких як керування запасами, система членства, запис постачальників, ведення бухгалтерського обліку, оформлення замовлень на купівлю, котирування та переміщення запасів, створення прихованих етикеток зі штрих-кодом, звітність про продажі.

Тим не менш, як серед кінцевих користувачів, так і серед постачальників, модним є термін POS-система, а не система управління роздрібною торгівлею.

Основне, фундаментальне визначення системи POS – це система, яка дозволяє обробляти та записувати транзакції між компанією та її споживачами під час придбання товарів та/або послуг.

Дизайн торгової вітрини є найважливішим для користувача. Цей інтерфейс користувача дуже важливий у порівнянні з інтерфейсом в інших пакетах програмного забезпечення, таких як текстові редактори або програми для роботи з електронними таблицями, де швидкість навігації не настільки важлива для продуктивності бізнесу.

Для підприємств, розташованих у найкращих місцях, де вартість нерухомості є високою, часто можна побачити чергу клієнтів. Чим швидше завершується продаж, тим коротший час у черзі, що підвищує задоволеність клієнтів, що вигідно покупцям і персоналу. Операції з високим трафіком, такі як продуктові магазини та кафе, мають швидко обробляти продажі на касі, тому потік інтерфейсу користувача часто розроблено з мінімальною кількістю спливаючих вікон або інших перерв, щоб гарантувати, що оператор не відволікається, і трансакція може бути оброблена так швидко, як можливо.

Незважаючи на те, що покращення ергономіки можливе, чистий, швидкий вигляд може відбуватися за рахунок принесення в жертву функцій, які часто потрібні кінцевим користувачам, таких як знижки, доступ до зароблених екранів, членство та схеми лояльності можуть передбачати перегляд інших функцій POS для того, щоб екран точки продажу містив лише те, що потрібно касиру для обслуговування клієнтів.

1.2. Особливості POS-рішень в сільськогосподарському секторі

POS-рішення (Point of Sale) в сільськогосподарському секторі мають свої особливості.

POS-системи в сільському господарстві повинні враховувати особливості цього сектора, такі як робота зі зважуванням продукції, обробка касових операцій для різних категорій товарів (наприклад, сільськогосподарські продукти, засоби захисту рослин тощо).

POS-рішення повинні мати можливість вести облік запасів сільськогосподарської продукції, включаючи керування постачанням і перевірку наявності товарів на складі.

POS-системи часто повинні взаємодіяти з іншими сільськогосподарськими програмами, такими як системи управління фермою, програми для обробки польових даних і т.д. Інтеграція дозволяє автоматизувати обмін даними і забезпечує точність і ефективність операцій.

Сільськогосподарські підприємства часто мають розподілену і мобільну робочу силу. Тому POS-системи повинні підтримувати мобільні пристрої, такі як планшети або смартфони, щоб працівники могли проводити операції на місці, наприклад, в полі або на ринку.

Сільськогосподарські підприємства потребують точних звітів та аналітики для управління бізнесом. POS-системи повинні надавати зручні інструменти для створення звітів про продажі, інвентаризацію, аналіз продуктивності тощо.

Ці особливості дозволяють сільськогосподарським підприємствам ефективно управляти продажами, запасами і забезпечувати точність обліку в контексті їхньої діяльності.

1.3. Технічне завдання

1. Загальні відомості

1.1. Найменування системи

1.1.1. Повне найменування системи – «SkyService»

1.1.2. Скорочене найменування системи – «SkyService»

1.2. Планові терміни початку та закінчення робіт

Дата початку: 01.01.2023

Дата закінчення: 01.06.2023

1.3. Порядок оформлення і пред’явлення результатів робіт

Завершення розробки системи POS-рішення, включаючи функціональність, інтерфейс користувача, базу даних та інші компоненти. Тестування системи для перевірки його правильності, надійності та забезпечення заданих функцій. Документування результатів робіт, включаючи технічну документацію, пояснювальні записки та інструкції користувача. Підготовка звіту про роботу, в якому будуть описані основні характеристики системи, результати тестування та інші важливі відомості. Пред'явлення результатів робіт замовнику або відповідним представникам підприємства для оцінки, перевірки та затвердження.

1.4. Головний бенефіціар та потенційні користувачі системи

Головним бенефіціаром системи POS-рішення для сільськогосподарського підприємства є саме підприємство, яке замовляє і використовує цю систему. Потенційні користувачі системи включають різні підрозділи та працівників підприємства, які займаються продажем і торговельною діяльністю. Це можуть бути менеджери магазину, касири, складські робітники та інші співробітники, які взаємодіють з системою POS-рішення для проведення операцій продажу, управління запасами та обробки платежів.

2. Мета та призначення створення системи

2.1. Призначення системи

Призначення системи POS-рішення для сільськогосподарського підприємства полягає в автоматизації процесів продажу та управління торговою діяльністю на підприємстві. Ця система дозволяє зручно та ефективно виконувати операції з продажу, управляти запасами товарів, обробляти платежі та забезпечувати точність обліку.

2.2. Мета створення системи

Метою створення системи POS-рішення для сільськогосподарського підприємства є поліпшення ефективності торговельних операцій та управління, зниження ризиків помилок та втрат, а також підвищення задоволення клієнтів. Система спрощує процес продажу, дозволяючи швидко виконувати операції та забезпечувати точність інформації, необхідної для прийняття управлінських рішень.

3. Вимоги до системи

3.1. Вимоги до системи в цілому

3.1.1. Вимоги до структури та функціонування системи, перелік підсистем

3.1.1.1. Вимоги до способів і засобів інформаційного обміну між компонентами системи

Система повинна підтримувати передачу даних між різними компонентами за допомогою стандартних протоколів комунікації, таких як TCP/IP або HTTP. Для безпеки обміну даними необхідно використовувати шифрування та механізми автентифікації. Система повинна забезпечувати можливість синхронного та асинхронного обміну даними в залежності від потреб користувача та специфіки операцій. Вимоги до пропускної здатності мережі повинні бути враховані для забезпечення швидкого та ефективного обміну даними.

3.1.1.2. Вимоги до режимів функціонування системи

Система повинна працювати у режимі онлайн, що дозволяє миттєво оновлювати дані про продажі, запаси та іншу інформацію. Надійність системи має бути високою, забезпечуючи стабільну та безперебійну роботу навіть при виникненні непередбачуваних ситуацій, таких як відмови обладнання чи перебої в електропостачанні. Вимоги до швидкості роботи системи повинні бути виконані, забезпечуючи миттєву відповідь на запити користувачів та оперативне виконання операцій продажу.

3.1.1.3. Вимоги до діагностування системи

Система повинна мати вбудовані механізми діагностики та моніторингу, що дозволяють виявляти та локалізувати помилки та несправності. Вимоги до журналювання дій користувачів та системних подій повинні бути виконані для подальшого аналізу та відновлення стану системи в разі необхідності.

3.1.1.4. Вимоги до режимів управління системою

Система повинна мати інтерфейс управління, що дозволяє адміністраторам налаштовувати параметри системи, додавати та видаляти користувачів, керувати рівнями доступу та інші налаштування. Забезпечення механізмів резервного копіювання та відновлення даних для забезпечення безпеки і можливості відновлення системи в разі втрати даних чи випадкових помилок. Вимоги до масштабованості системи повинні бути виконані, щоб забезпечити можливість розширення та адаптації системи під зростаючі потреби підприємства.

3.1.2. Показники призначення

3.1.2.1. Параметри, що характеризують ступінь відповідності системи призначенням

Функціональність - система повинна задовольняти основні потреби сільськогосподарського підприємства в автоматизації процесів продажу, управління запасами та обробки платежів.

Сумісність - система повинна бути сумісною з існуючими інформаційними системами та обладнанням, які використовуються на підприємстві.

Надійність - система повинна працювати стабільно та безперебійно, забезпечуючи доступ до необхідної інформації у будь-який час.

Продуктивність - система повинна забезпечувати достатню швидкість та продуктивність для виконання операцій продажу в режимі реального часу.

3.1.2.2. Вимоги до пристосовності системи до змін

Модульність - система повинна бути побудована з окремих модулів, що дозволяють легко вносити зміни або додавати нові функціональні можливості без значного впливу на роботу інших компонентів.

Гнучкість - система повинна мати гнучку архітектуру, що дозволяє адаптуватися до змін у вимогах та процесах сільськогосподарського підприємства.

Легкість розширення - система повинна бути легко розширюваною, щоб враховувати зростаючі потреби та розширення діяльності підприємства.

3.1.2.3. Вимоги до збереження працездатності системи в різних ймовірних умовах

Резервне копіювання - система повинна мати механізми регулярного резервного копіювання даних для запобігання втраті важливої інформації.

Відновлення після відмови - система повинна мати можливість швидкого відновлення після відмови апаратного чи програмного забезпечення.

Захист від збоїв електропостачання - система повинна бути захищена від можливих перебоїв в електропостачанні шляхом використання UPS або інших джерел резервного живлення.

3.1.3. Вимоги до надійності

3.1.3.1. Склад показників надійності до системи в цілому

Система повинна забезпечувати стабільну та безперебійну роботу протягом тривалого періоду без несправностей чи відмов. Система повинна мати механізми швидкого відновлення після відмови, щоб забезпечити мінімальний час простою та перерв у роботі. Система повинна бути стійкою до непередбачуваних ситуацій, помилок апаратного чи програмного забезпечення та здатною до нормальної роботи навіть у разі виникнення проблем. Система повинна мати механізми збереження та відновлення даних, щоб запобігти втраті важливої інформації.

3.1.3.2. Вимоги до надійності технічних засобів і програмного забезпечення

Технічні засоби, які використовуються в системі, повинні бути стабільними та надійними, з високою тривалістю безвідмовної роботи. Програмне забезпечення системи повинно бути надійним, без помилок та дефектів, що можуть призвести до неправильної роботи або збоїв. Система повинна мати вбудовані механізми захисту від злому та кібератак, щоб забезпечити конфіденційність, цілісність та доступність даних.

3.1.3.3. Вимоги до методів оцінки і контролю показників надійності на різних стадіях створення системи

Система повинна пройти обов'язкові етапи тестування, включаючи модульне тестування, інтеграційне тестування та приймальне тестування, для перевірки надійності та відповідності вимогам. Система повинна мати механізми моніторингу, що дозволяють відстежувати та контролювати роботу системи, виявляти можливі проблеми та реагувати на них вчасно. Система повинна підтримувати аудит безпеки, щоб виявляти потенційні вразливості та неправильні налаштування, які можуть вплинути на надійність системи.

3.1.4. Вимоги до ергономіки та технічної естетики

Система повинна мати інтуїтивно зрозумілий та зручний інтерфейс, що дозволяє операторам легко виконувати роботу та зменшує ймовірність помилок. Дизайн системи повинен бути привабливим та естетичним, створюючи позитивне враження у користувачів. Фізичне обладнання, таке як касові апарати та сканери, повинно бути зручним у використанні та адаптованим до потреб користувачів.

3.1.5. Вимоги до експлуатації, технічного обслуговування, ремонту і зберігання компонентів системи

Система повинна бути простою у використанні, з мінімальною потребою в спеціалізованій підготовці операторів. Компоненти системи повинні мати можливість регулярного технічного обслуговування, включаючи оновлення програмного забезпечення та встановлення оновлень без перерв у роботі. Система повинна бути піддається ремонту та заміні окремих компонентів без суттєвого впливу на загальну функціональність. Компоненти системи повинні мати вимоги до умов зберігання, включаючи температуру, вологість та захист від пилу та інших шкідливих факторів.

3.1.6 Вимоги до захисту інформації від несанкціонованого доступу

3.1.6.1. Вимоги до інформаційної безпеки

Конфіденційність даних. Система повинна забезпечувати захист конфіденційної інформації, такої як фінансові дані, персональні дані клієнтів та інші конфіденційні відомості.

Цілісність даних. Система повинна гарантувати, що дані не підлягають неправомірним змінам або втраті внаслідок несанкціонованого доступу, помилок або збоїв системи.

Доступність. Система повинна бути доступною для авторизованих користувачів відповідно до їх ролей та повноважень, забезпечуючи неперервний доступ до необхідної функціональності.

Аудит безпеки. Система повинна здійснювати аудит безпеки, що включає реєстрацію подій, виявлення можливих загроз та незвичайних активностей, а також вжиття заходів для їхнього усунення.

3.1.6.2. Вимоги до антивірусного захисту

Система повинна мати встановлене актуальне антивірусне програмне забезпечення для виявлення та запобігання інфікуванню шкідливими програмами. Антивірусне програмне забезпечення повинно мати механізми автоматичного оновлення вірусних баз даних для виявлення нових загроз та ефективного виявлення інфікованого програмного забезпечення. Система повинна піддаватися регулярному скануванню антивірусним програмним забезпеченням для виявлення та усунення потенційно небезпечних елементів. Система повинна мати механізми для блокування доступу до шкідливих веб-сайтів, які можуть становити загрозу для безпеки.

3.1.6.3. Розмежування відповідальності ролей при доступі

Система повинна мати механізми автентифікації користувачів та надання їм відповідних прав доступу на основі їхніх ролей та повноважень. Користувачам повинно бути надано доступ лише до необхідних функціональних можливостей та даних, відповідно до їхніх обов'язків та відповідальності. Система повинна вести журнал доступу, що реєструє дії користувачів, забезпечуючи можливість перевірки та виявлення можливих некоректних дій або порушень безпеки.

3.1.7. Вимоги до захисту від впливу зовнішніх факторів

Система повинна бути захищена від небезпек, таких як урагани, повені, пожежі, фізичні пошкодження або крадіжки, шляхом встановлення безпечного фізичного середовища, використання захисного обладнання та резервного копіювання даних.

Система повинна мати заходи безпеки, щоб запобігти несанкціонованому доступу, хакерським атакам, витоку даних, вірусам та іншим кіберзагрозам.

Система повинна мати механізми для регулярного резервного копіювання даних, щоб запобігти втраті інформації у разі непередбачених ситуацій.

3.1.8. Вимоги безпеки

Система повинна мати механізми для перевірки ідентифікації користувачів та надання їм відповідних прав доступу. Конфіденційна інформація, передана через систему, повинна бути зашифрована для запобігання несанкціонованого доступу до неї. Система повинна мати механізми для моніторингу безпеки, виявлення підозрілих активностей та сповіщення про можливі загрози. Система повинна мати можливість оновлювати безпекові заходи, включаючи встановлення патчів та оновлення програмного забезпечення для запобігання використанню вразливостей. Система повинна мати заходи для запобігання витоку конфіденційної інформації, включаючи контроль доступу та шифрування даних.

3.2. Перелік підсистем системи (при наявності підсистем).

Підсистема управління продажами. Включає функції для обробки замовлень, видачі чеків, керування запасами та інші операції, пов'язані з продажами.

Підсистема управління складом. Забезпечує контроль за запасами товарів, відстеження постачань, приймання та відвантаження товарів на складі.

Підсистема звітності та аналітики. Надає можливість генерувати різноманітні звіти про продажі, стан запасів, фінансові показники та інші дані, необхідні для аналізу та прийняття управлінських рішень.

Підсистема обліку фінансів. Включає функціональність для обліку грошових операцій, оплати рахунків, ведення касових операцій та бухгалтерського обліку.

Підсистема управління клієнтами. Дозволяє вести базу даних клієнтів, збирати та аналізувати дані про них, надавати програми лояльності та інші сервіси для покращення взаємодії з клієнтами.

3.3. Вимоги до видів забезпечення

3.3.1. Вимоги до математичного забезпечення

Система повинна мати здатність до математичного аналізу даних, включаючи розрахунки, статистичний аналіз, моделювання та інші математичні операції, необхідні для здійснення аналітичних процесів та прийняття управлінських рішень. Система повинна мати можливість виконувати математичні розрахунки, пов'язані з фінансовими показниками, такими як оборотність запасів, рентабельність, податкові розрахунки та інші фінансові обчислення. Система може включати математичні алгоритми та моделі для прогнозування попиту, оптимізації розкладу поставок, планування виробництва та інших операційних задач, що допомагають вдосконалити ефективність діяльності підприємства. У випадку, коли POS-рішення використовується для сільськогосподарського підприємства, система може мати математичні моделі, що дозволяють прогнозувати та оптимізувати ріст рослин, враховуючи фактори, такі як погода, ґрунтові умови, внесення добрив та інші аспекти, що впливають на сільськогосподарські процеси. Система може містити математичні алгоритми для калькуляцій вартості продукції, ціноутворення, розрахунку маржі та прибутку, що допомагають забезпечити оптимальну стратегію ціноутворення для підприємства.

3.3.2. Вимоги до інформаційного забезпечення

3.3.2.1. Вимоги до складу, структури і способів організації даних в системі

Склад. Система повинна забезпечувати належний набір даних, необхідних для ефективного виконання функцій POS-рішення, таких як інформація про продукти, клієнтів, склади, транзакції та інші релевантні дані.

Структура. Дані повинні бути організовані у структурованому форматі, що дозволяє зручний доступ, пошук та обробку інформації. Наприклад, можуть використовуватись бази даних з відповідними таблицями, схемами та зв'язками між ними.

Способи організації даних. Система повинна мати визначені методи і протоколи для збереження, оновлення та видалення даних. Наприклад, можуть використовуватись механізми резервного копіювання, контролю версій, аудиту доступу та інші методи забезпечення цілісності та доступності даних.

3.3.2.2. Вимоги до інформаційного обміну між компонентами системи

Система повинна визначати стандартні формати та протоколи для передачі і обміну даними між різними компонентами, наприклад, через веб-сервіси або API.

При передачі даних між компонентами системи необхідно забезпечити їхню цілісність і безпеку, наприклад, шляхом використання криптографічних методів шифрування та підпису.

Система повинна забезпечувати ефективний та надійний обмін даними між компонентами для забезпечення швидкої та безперебійної роботи POS-рішення.

3.3.2.3. Вимоги до інформаційної сумісності із суміжними системами

POS-рішення повинно бути сумісним та інтегровано з іншими системами, що використовуються у сільськогосподарському підприємстві, наприклад, системами управління запасами, бухгалтерії, управління

персоналом тощо. Дані між POS-рішенням та суміжними системами повинні бути синхронізовані для забезпечення єдиної та актуальної інформації в усіх системах.

3.3.2.4. Вимоги використання класифікаторів та уніфікованих документів

Система повинна підтримувати використання стандартних класифікаторів та уніфікованих документів, що використовуються в сільськогосподарському секторі, для забезпечення однорідності та зручності обробки даних.

Система може забезпечувати автоматичне заповнення документів на основі введених даних, що спрощує процес роботи з документацією.

3.3.2.5. Вимоги щодо застосування систем управління базами даних

Використання СУБД. Система повинна використовувати систему управління базами даних (СУБД) для ефективного зберігання, керування та доступу до даних.

Надійність та масштабованість. СУБД повинна бути надійною, забезпечувати захист даних та бути здатною обробляти великий обсяг інформації в реальному часі.

3.3.2.6. Вимоги до структури процесу збору, обробки, передачі даних в системі представлення даних

Система повинна мати механізми для збору даних з різних джерел, таких як сканери, датчики, бази даних тощо.

Система повинна забезпечувати обробку та аналіз зібраних даних для отримання важливої інформації, наприклад, статистичних даних про продажі, запаси тощо.

Система повинна мати механізми передачі даних до інших компонентів системи або зовнішніх систем, забезпечуючи безпеку та цілісність даних.

3.3.2.7. Вимоги до захисту даних від руйнувань при аваріях і збоях в електроживленні системи

Система повинна мати механізми регулярного резервного копіювання даних для відновлення інформації в разі аварій або збоїв та заходи захисту від відключення електроживлення, такі як використання резервних джерел живлення або UPS.

3.3.2.8. Вимоги до контролю, зберіганню, оновленню та відновленню даних

Система повинна мати механізми контролю доступу до даних, що забезпечують обмеження прав доступу та захист від несанкціонованого використання.

Система повинна забезпечувати безпечне зберігання даних, а також механізми оновлення та синхронізації даних для підтримки актуальності інформації.

У разі втрати або пошкодження даних система повинна мати механізми для їхнього відновлення з резервних копій або інших джерел.

3.3.2.9. Вимоги до процедури надання юридичної сили документам, що продукуються технічними засобами системи

Система повинна забезпечувати автентичність документів, що продукуються, шляхом застосування цифрових підписів або інших методів ідентифікації та перевірки автентичності.

Система повинна бути інтегрована з відповідними юридичними системами, які регулюють процедуру надання юридичної сили документам. Це може включати забезпечення відповідності стандартам електронного документообігу та використання визнаних юридичних форматів.

Система повинна забезпечувати довготривале зберігання документів з юридичною силою відповідно до вимог законодавства. Це може включати

застосування методів архівування, резервного копіювання та забезпечення надійності збереження даних.

Система повинна забезпечувати можливість відтворення документів з юридичною силою у вихідному форматі, який зберігає інформацію в незмінному вигляді.

Система повинна мати механізми аудиту та журналювання, що дозволяють відстежувати всі дії, пов'язані з процедурою надання юридичної сили документам. Це допоможе забезпечити відповідність процедури вимогам законодавства та відповідності юридичним нормам.

Система повинна мати механізми контролю доступу, що забезпечують обмежений доступ до документів з юридичною силою, залежно від ролей та прав доступу користувачів.

Користувачі системи повинні бути належним чином інформовані про процедуру надання юридичної сили документам технічними засобами системи, а також про обов'язки та відповідальність, пов'язані з використанням таких документів.

4. Вимоги до програмного забезпечення

Система повинна мати можливість обробляти транзакції купівлі-продажу, вести облік складів та запасів, забезпечувати звітність та аналітику, керувати клієнтською базою даних, підтримувати різні види платежів та інші необхідні функції для ефективної роботи сільськогосподарського підприємства. Система повинна мати зручний та інтуїтивно зрозумілий інтерфейс, що дозволяє просте та зручне використання POS-рішення, навіть користувачам без попереднього досвіду. Система повинна бути надійною, забезпечувати захист даних та запобігати несанкціонованому доступу до системи та конфіденційної інформації. Система повинна бути сумісною з

різними операційними системами, базами даних та апаратним забезпеченням, що використовується в сільськогосподарському підприємстві.

5. Вимоги до технічного забезпечення

Система вимагає наявності комп'ютерів або інших пристроїв з достатньою продуктивністю та пам'яттю для забезпечення швидкості та ефективності роботи. Для POS-рішення необхідна наявність мережевого з'єднання для забезпечення зв'язку між компонентами системи та можливості обміну даними. Для друку чеків та сканування штрих-кодів необхідні відповідні принтери та сканери.

6. Вимоги до методичного забезпечення

Користувачі системи повинні мати доступ до навчальних матеріалів, посібників та тренінгів для ознайомлення з функціоналом та правилами використання POS-рішення. Повинна бути надана технічна підтримка для вирішення будь-яких проблем, пов'язаних з роботою системи, та для надання консультацій користувачам. Система повинна мати документацію, яка описує процеси встановлення, налаштування та використання POS-рішення, а також інструкції з обслуговування та усунення неполадок.

1.4. Висновок до розділу 1

POS-рішення в сільськогосподарському секторі мають свої особливості, які враховують специфіку галузі, зокрема зважування продукції і обробка касових операцій для різних категорій товарів. Важливо, щоб POS-системи в сільському господарстві могли керувати запасами сільськогосподарської продукції, включаючи постачання і перевірку наявності товарів на складі. Інтеграція з іншими сільськогосподарськими системами дозволяє автоматизувати обмін даними і підвищує точність та ефективність операцій.

Підтримка мобільних пристроїв дозволяє працівникам проводити операції на місці, що особливо важливо для розподіленої робочої сили в сільському господарстві. POS-системи повинні надавати зручні засоби для створення звітів про продажі, інвентаризацію та аналіз продуктивності, що допомагає управляти бізнесом.

Ці особливості допомагають сільськогосподарським підприємствам підвищити ефективність операцій, забезпечити точність обліку і управляти своїм бізнесом з використанням POS-рішень. POS-рішення призначене для забезпечення ефективного управління продажами та обліку в сільськогосподарському підприємстві. Основною метою створення POS-рішення є поліпшення процесу продажів, оптимізація обліку товарів, автоматизація фінансових операцій та забезпечення точності та надійності обробки даних.

Система повинна забезпечувати ефективний і безперебійний інформаційний обмін між компонентами системи, режими функціонування, діагностику та управління системою. Система повинна мати високу ступінь відповідності своєму призначенню та бути гнучкою для змін в сільськогосподарському середовищі, повинна бути стійкою до різних умов експлуатації та забезпечувати надійність технічних засобів і програмного забезпечення.

Система повинна відповідати принципам ергономіки та технічної естетики для забезпечення зручного та приємного користувацького досвіду, мати вимоги до експлуатації, технічного обслуговування, ремонту і зберігання компонентів системи, щоб забезпечити їхню тривалу та надійну роботу.

РОЗДІЛ 2
ВИБІР ПРОГРАМНИХ ЗАСОБІВ ТА ПРОЄКТУВАННЯ ПРОГРАМНОЇ КОМПОНЕНТИ

2.1. Чинники, що визначають хід і результати роботи програмної компоненти POS-рішення

Чинники, що визначають хід і результати роботи програмної компоненти POS-рішення, включають наступне (див. рис. 2.1.):

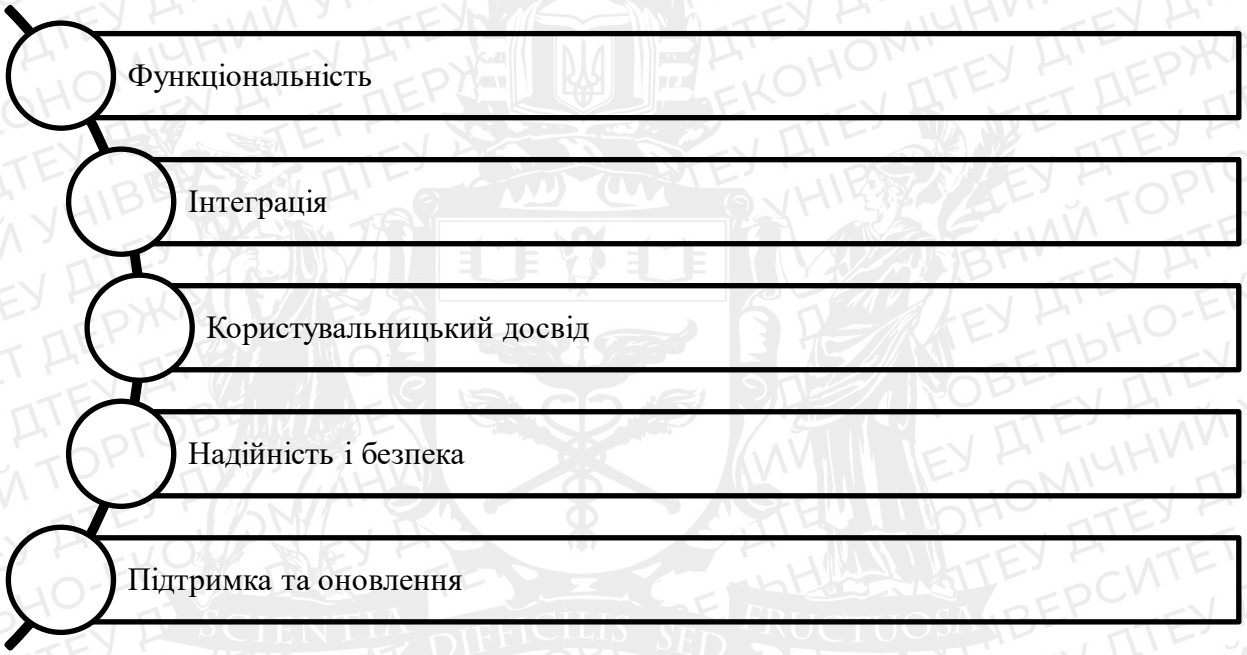

Рис. 2.1. Чинники, що визначають хід і результати роботи програмної компоненти POS-рішення

Якість і розмаїття функцій, які надає програмна компонента POS-рішення, впливають на її здатність задовольнити потреби бізнесу.

Основні функції, які можуть бути включені в POS-рішення для сільськогосподарського сектору, включають (див. рис. 2.2.):

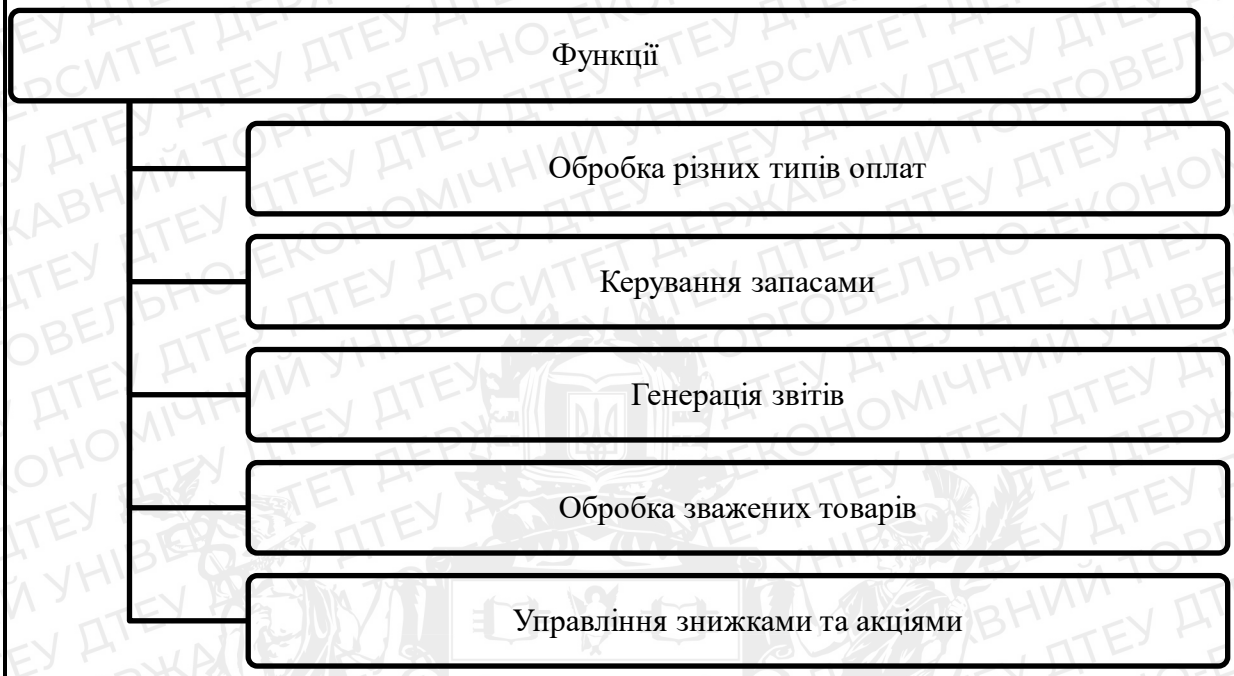

Рис. 2.2. Основні функції POS-рішення для сільськогосподарського сектору

1. Обробка різних типів оплат. Добре розроблене POS-рішення повинно забезпечувати можливість обробки різних типів оплат, таких як готівка, кредитні та дебетові картки, безконтактні платежі тощо. Воно повинно бути сумісним з різними платіжними шлюзами та платіжними системами.

2. Керування запасами. POS-система повинна мати функції керування запасами, що дозволяють вести облік товарів і матеріалів. Це включає в себе стеження за наявністю товарів на складі, автоматичне оновлення запасів при продажу і поповнення запасів при поставках.

3. Генерація звітів. POS-рішення повинно надавати можливість генерувати різні звіти, такі як звіти про продажі, звіти про запаси, фінансові звіти тощо. Це дозволяє бізнесу отримувати необхідну інформацію про його продуктивність, стан запасів і фінансові результати.

4. Обробка зважених товарів. У сільському господарстві можуть продаватися продукти, що вимагають зважування, наприклад, фрукти, овочі

або м'ясо. Тому POS-рішення повинно мати можливість точного зважування товарів і розрахунку їх вартості з урахуванням ваги.

5. Управління знижками та акціями. POS-система може включати функціонал для керування знижками, акціями та програмами лояльності. Це дозволяє бізнесу пропонувати спеціальні умови продажу та залучати клієнтів.

Комбінація цих функцій у програмній компоненті POS-рішення дозволяє сільськогосподарським підприємствам ефективно управляти продажами, контролювати запаси і отримувати необхідну аналітику для прийняття рішень.

Інтеграція є важливим фактором в роботі POS-рішення в сільськогосподарському секторі, оскільки вона дозволяє забезпечити безперебійний обмін даними та автоматизувати багато процесів.

POS-рішення повинно бути здатним інтегруватися з бухгалтерськими програмами, такими як системи обліку доходів і витрат, програми оподаткування і фінансового обліку. Це дозволяє автоматично передавати дані про продажі, оплату, знижки та податки з POS-системи до бухгалтерської системи, спрощуючи процес обліку та звітності.

Успішне POS-рішення повинно інтегруватися з системами управління запасами, що дозволяє автоматично оновлювати і синхронізувати дані про наявність товарів на складі. Наприклад, при продажу товару через POS-систему, запаси автоматично зменшуються, що дозволяє вести точний облік запасів і уникнути нестачі або перевищення.

Для зручності клієнтів і забезпечення безпеки платежів, POS-рішення повинно бути сумісним з різними електронними платіжними шлюзами. Це дозволяє здійснювати швидкі та безпечні онлайн-платежі з використанням різних платіжних каналів.

Для додаткової функціональності та розширених можливостей, POS-рішення може бути інтегровано зі зовнішніми системами через відкриті API (інтерфейс програмування додатків). Наприклад, це може бути інтеграція з

системами управління клієнтськими базами даних, системами електронного документообігу або системами управління виробництвом.

Користувальницький досвід є важливим аспектом POS-рішення, оскільки він безпосередньо впливає на продуктивність та ефективність роботи користувачів.

POS-система повинна мати зрозумілий та логічно організований інтерфейс, який легко засвоюється користувачами. Це включає чітку навігацію, легкість у використанні основних функцій, зрозумілі інструкції та інтуїтивне розміщення елементів і кнопок. POS-рішення повинно забезпечувати швидкий доступ до необхідної інформації, такої як ціни товарів, наявність на складі, знижки та акції. Користувачі повинні мати можливість швидко знайти потрібну інформацію без зайвих зусиль.

POS-система повинна бути простою у використанні, навіть для користувачів без попереднього досвіду. Інтуїтивність та простота роботи з програмою допомагають зменшити час навчання і підвищити продуктивність праці. Користувачі повинні мати можливість настроювати POS-систему відповідно до своїх потреб і вимог. Це включає можливість настроювати меню, розміщення елементів і кнопок, налаштування звітів та інші параметри, щоб система оптимально відповідала їхньому бізнесу.

Якщо POS-рішення надає відповідну підтримку та навчання користувачам, це сприяє полегшенню процесу впровадження та забезпеченню ефективного використання системи. Підтримка може включати документацію, онлайн-ресурси, навчальні матеріали та можливість звернутися до технічної підтримки у разі потреби.

Таким чином, хороший користувальницький досвід допомагає покращити продуктивність роботи з POS-рішенням, зменшити помилки та забезпечити задоволення користувачів від використання системи.

Надійність і безпека є ключовими аспектами POS-рішень, особливо у сільськогосподарському секторі, де велике значення мають надійність системи

та захист конфіденційної і критичної інформації. POS-рішення повинні бути стабільними та працездатними, щоб уникнути перебоїв у роботі, особливо під час проведення операцій продажу та обробки платежів. Це означає, що програмна компонента повинна бути розроблена з використанням надійних технологій, міцної архітектури та добре протестована перед впровадженням. POS-системи повинні забезпечувати високий рівень захисту конфіденційної інформації, такої як персональні дані клієнтів, фінансові дані та інші критичні дані. Це може включати шифрування даних, застосування безпечних протоколів передачі даних, захист від несанкціонованого доступу та зловживань. Крім того, важливо встановити процедури резервного копіювання та відновлення даних для уникнення втрати даних у разі випадкової або системної помилки. OS-рішення повинні мати вбудовані механізми для виявлення та запобігання шахрайству, таким як використання функцій аутентифікації, перевірка платежів та виявлення підозрілих транзакцій. Це допомагає уникнути втрати коштів і забезпечити безпеку бізнесу та його клієнтів. POS-системи повинні мати механізми резервного копіювання та відновлення даних для забезпечення безперебійної роботи у разі випадків аварійного відключення або втрати даних. Це дозволяє швидко відновити роботу системи та запобігти втраті даних і призупиненню бізнес-процесів.

Загальна надійність і безпека POS-рішення відіграють критичну роль у забезпеченні незавершенної роботи та захисту важливих даних сільськогосподарських підприємств. Правильна розробка, використання надійних технологій та впровадження заходів безпеки допомагають підвищити довіру до системи та забезпечити стабільну та безпечну роботу.

Наявність якісної технічної підтримки від постачальника POS-рішення є важливою. Користувачі повинні мати можливість звернутися до фахівців з питань налаштування, усунення несправностей та вирішення технічних

проблем. Швидкість та якість відповіді з боку служби підтримки можуть значно вплинути на продуктивність бізнесу.

Постачальник POS-рішення повинен регулярно надавати оновлення програмного забезпечення з метою вдосконалення функціональності, виправлення помилок та забезпечення безпеки. Це включає патчі, оновлення версій програми та доступ до нових функціональних можливостей. Наявність активного циклу оновлень дозволяє підтримувати POS-систему в актуальному та ефективному стані.

Постачальник POS-рішення повинен надавати навчання користувачам щодо роботи з програмною компонентою. Це може бути виготовлення навчальних матеріалів, вебінарів, особистого тренінгу та підтримки під час впровадження системи. Правильне навчання допомагає користувачам швидше освоїти систему та ефективно використовувати її функціонал.

Якість підтримки та оновлень впливає на задоволення клієнтів. Постачальник POS-рішення повинен бути відкритим до отримання зворотного зв'язку та готовим враховувати потреби користувачів для поліпшення свого продукту.

Ефективна підтримка та надання оновлень допомагають підтримувати POS-систему у високопродуктивному та актуальному стані, забезпечують безперебійну роботу та задоволення користувачів.

Успішна робота POS-рішення залежить від цих факторів, і їх оптимальне впровадження сприяє покращенню ефективності та результативності бізнесу.

2.2. Інструменти розробки програмної компоненти POS-рішення

Існує багато підходів до оцінки методів розробки ПЗ. З одного боку, використовуються досить стандартні основи для оцінки. Завдяки своєму загальному характеру ця структура підходить для оцінки конкретних питань в компонентно-орієнтованих методах.

Основні інструменти, які можуть бути використані при розробці програмної компоненти POS-рішення наведені на рис. 2.3.

Рис. 2.3. Інструменти розробки програмної компоненти POS-рішення

Розглянемо усі ці інструменти більш детально.

Мови програмування є ключовим елементом в розробці програмної компоненти POS-рішення. Основна мова програмування може бути обрана залежно від вимог і технічних уподобань.

Java є широко використовуваною мовою програмування для розробки POS-рішень. Вона пропонує потужність, надійність та переносимість між різними платформами. З використанням Java можна розробляти як серверну частину POS-рішення, так і клієнтські додатки.

C# є мовою програмування, яка широко використовується для розробки програмного забезпечення під платформу .NET. Вона має потужні функціональні можливості і гармонійно інтегрується з іншими технологіями Microsoft, що може бути корисним для POS-рішень, особливо якщо ви плануєте інтегрувати їх з Windows-середовищем.

Python - це проста і легко засвоювана мова програмування, яка набула великої популярності в галузі розробки. Вона має широкий набір бібліотек і фреймворків, що полегшує розробку POS-рішень. Python також має підтримку різних платформ, що дає можливість розгортати POS-додатки на різних операційних системах.

JavaScript є мовою програмування, яка використовується для веб-розробки і може бути використана для створення веб-орієнтованих компонент POS-рішення. З його допомогою можна створювати динамічні інтерфейси користувача та взаємодіяти з веб-серверами.

Інтегроване середовище розробки (IDE) є важливим інструментом для розробки програмної компоненти POS-рішення. Воно надає зручність і продуктивність, сприяє швидкому написанню, налагодженню і тестуванню коду.

Eclipse є одним з найпопулярніших IDE для розробки різних типів додатків, включаючи Java-програми. Він надає розширені можливості редагування коду, налагодження, автоматичного доповнення і підтримки версій, що полегшує розробку POS-рішень.

Visual Studio - це популярне IDE, розроблене компанією Microsoft для розробки програмного забезпечення на платформі .NET. Воно надає широкі можливості для розробки додатків, включаючи POS-рішення, з використанням мови програмування C# і платформи .NET.

PyCharm є спеціалізованим IDE для розробки проектів на мові програмування Python. Воно надає потужні інструменти для автоматичного доповнення коду, налагодження, тестування і аналізу коду, що допомагає розробникам Python ефективно створювати POS-рішення.

Visual Studio Code (VS Code) є легковаговим, розширюваним редактором коду, який підтримує багато мов програмування і фреймворків. Він надає потужні можливості редагування, налагодження, тестування і інтеграції з іншими інструментами, що робить його привабливим для розробки POS-рішень.

Кожен з цих IDE має свої особливості, і вибір залежить від уподобань, мов програмування і платформи, на якій планується розробляти POS-рішення.

Фреймворки грають важливу роль в розробці програмної компоненти POS-рішення, оскільки вони надають готові компоненти, бібліотеки та

рішення для спрощення розробки і підтримки основних функціональних можливостей. Ось кілька популярних фреймворків, які можна використовувати для розробки POS-рішень у різних мовах програмування (див.. табл. 2.1.):

Таблиця 2.1

Фреймворки як інструменти для розробки POS-рішень

Фреймворк	Характеристика
	Java
Spring	Spring є одним з найпопулярніших фреймворків для розробки програмного забезпечення на Java. Він надає інверсію керування, аспектно-орієнтоване програмування, веб-фреймворк для створення веб-додатків та багато інших модулів, які полегшують розробку POS-рішень.
Hibernate	Hibernate є ORM (Object-Relational Mapping) фреймворком для роботи з базами даних у Java. Він дозволяє зручно працювати з реляційними базами даних, забезпечує мапування об'єктів на таблиці бази даних і спрощує операції збереження, оновлення та отримання даних.
	C#
ASP.NET	ASP.NET є фреймворком для розробки веб-додатків на платформі .NET. Використовуючи ASP.NET, можна створювати потужні веб-інтерфейси для POS-рішень, використовуючи мову програмування C#.
Entity Framework	Entity Framework є ORM фреймворком для роботи з базами даних у C#. Він надає простий і зручний спосіб взаємодії з реляційними базами даних, забезпечує мапування об'єктів на таблиці бази даних і автоматично генерує SQL-запити.
	Python
Django	Django є повнофункціональним веб-фреймворком для Python, який допомагає швидко розробляти веб-додатки. Він надає готові рішення для роботи з базами даних, URL-маршрутизації, автентифікації користувачів та іншого, що може бути корисним для розробки POS-рішень.
Flask	Flask є легким мікрофреймворком для Python, який дозволяє швидко створювати веб-додатки. Він має малу кількість залежностей і простоту використання, що робить його популярним для розробки простих POS-рішень.

База даних є важливою складовою для зберігання даних в POS-рішенні. Залежно від потреб і вимог проекту, використовуються різні типи баз даних. MySQL є одним з найпопулярніших відкритих реляційних баз даних. Він надає

широкі можливості для зберігання, керування і отримання даних. MySQL підтримує SQL-мову запитів і може бути легко інтегрований з багатьма різними мовами програмування. PostgreSQL є потужною реляційною базою даних з великою кількістю функцій і розширень. Вона підтримує розширений SQL, ACID-властивості і можливості геопросторового пошуку. PostgreSQL також є відкритою системою і може бути інтегрований з багатьма мовами програмування. Microsoft SQL Server є комерційною реляційною базою даних, яка надає різноманітні можливості для зберігання, керування і обробки даних. Вона підтримує розширений T-SQL, включає в себе інструменти для адміністрування та моніторингу баз даних, і є добре інтегрованою з екосистемою Microsoft. MongoDB є документ-орієнтованою нереляційною базою даних, яка надає гнучкість в роботі з неструктурованими даними. Вона зберігає дані у вигляді JSON-подібних документів і надає можливості горизонтального масштабування і реплікації. Redis є ключ-значення нереляційною базою даних, яка використовується для швидкого зберігання і отримання даних у пам'яті. Вона надає швидкий доступ до даних і підтримує різні типи даних, такі як рядки, хеші, списки і множини.

Вибір бази даних залежить від ваших потреб щодо схеми даних, обсягу і швидкодії, а також від підтримки мов програмування та інших інтеграційних можливостей, необхідних для POS-рішення.

HTML є основною мовою для створення структури і контенту веб-сторінок. Використовуючи HTML, ви можете визначити різні елементи, такі як текст, заголовки, таблиці, форми та інші, які будуть відображатися у веб-інтерфейсі POS-рішення.

CSS використовується для визначення зовнішнього вигляду і стилізації веб-сторінок. З його допомогою можна встановлювати кольори, шрифти, розміри, межі, фонові зображення та інші властивості елементів, що дозволяє створювати привабливий інтерфейс для POS-рішення.

API (Application Programming Interface) і бібліотеки є важливими компонентами для взаємодії з різними аспектами POS-системи і інтеграції зі зовнішніми сервісами.

Для забезпечення якості програмної компоненти POS-рішення використання інструментів тестування є важливим етапом. Юніт-тести використовуються для тестування окремих функцій або модулів програмного коду. Вони перевіряють, чи працюють окремі частини коду правильно і відповідають очікуваному результату. Деякі популярні фреймворки для написання юніт-тестів включають JUnit для Java, NUnit для C# і pytest для Python. Інтеграційні тести перевіряють взаємодію між різними компонентами POS-рішення. Вони переконуються, що всі частини працюють разом правильно і здатні обмінюватися даними та інформацією. Інструменти для автоматизації інтеграційних тестів, такі як Apache JMeter або Postman, можуть бути використані для створення тестових сценаріїв та перевірки взаємодії.

Функціональні тести перевіряють, чи працюють POS-функції і процеси відповідно до очікуваного функціоналу. Вони переконуються, що система виконує необхідні дії і повертає очікувані результати. Інструменти для автоматизації функціональних тестів, такі як Selenium WebDriver або Appium, можуть бути використані для створення тестових сценаріїв та виконання тестування.

2.3. Висновок до розділу 2

Java, C#, Python і JavaScript є популярними мовами для розробки POS-рішень. IDE допомагають створювати, налагоджувати і тестувати код. Приклади популярних IDE включають Eclipse, Visual Studio, PyCharm і Visual Studio Code. Використання фреймворків спрощує розробку і забезпечує швидку реалізацію функціональності. Наприклад, Spring, Hibernate, ASP.NET, Entity Framework, Django і Flask є популярними фреймворками для розробки POS-рішень.

Для зберігання даних POS-рішення можна використовувати реляційні бази даних, такі як MySQL, PostgreSQL або Microsoft SQL Server. Нереляційні бази даних, такі як MongoDB або Redis, також можуть бути використані. Для розробки веб-орієнтованої компоненти POS-рішення використовуються HTML, CSS і JavaScript. Фреймворки, такі як Angular, React і Vue.js, допомагають створювати потужні користувацькі інтерфейси.

Взаємодія з різними аспектами POS-системи вимагає використання API і бібліотек. Платіжні шлюзи, такі як Stripe або PayPal, можуть бути використані для обробки платежів. Інструменти тестування, такі як JUnit, NUnit і pytest, допомагають забезпечити якість програмної компоненти POS-рішення.

РОЗДІЛ 3

РОЗРОБКА ТА ВПРОВАДЖЕННЯ ПРОГРАМНОЇ КОМПОНЕНТИ POS-РІШЕННЯ «SKYSERVICE»

3.1. Функціональні характеристики розробленої програмної компоненти

Нами була розроблена програмна компонента POS-рішення "SkyService" для сільськогосподарського підприємства (див. рис. 3.1.). Вона буде використовуватися сільськосподарськими підприємствами для покращення операційних процесів організації, що значно пришвидшить роботу та підвищить її ефективність.

Функціональні характеристики компоненти наступні.

Програмна компонента дозволяє здійснювати продажі товарів і послуг на сільськогосподарському підприємстві. Вона надає можливість сканування або введення товарів, розрахунок суми покупки, прийом платежів і видачу здачі.

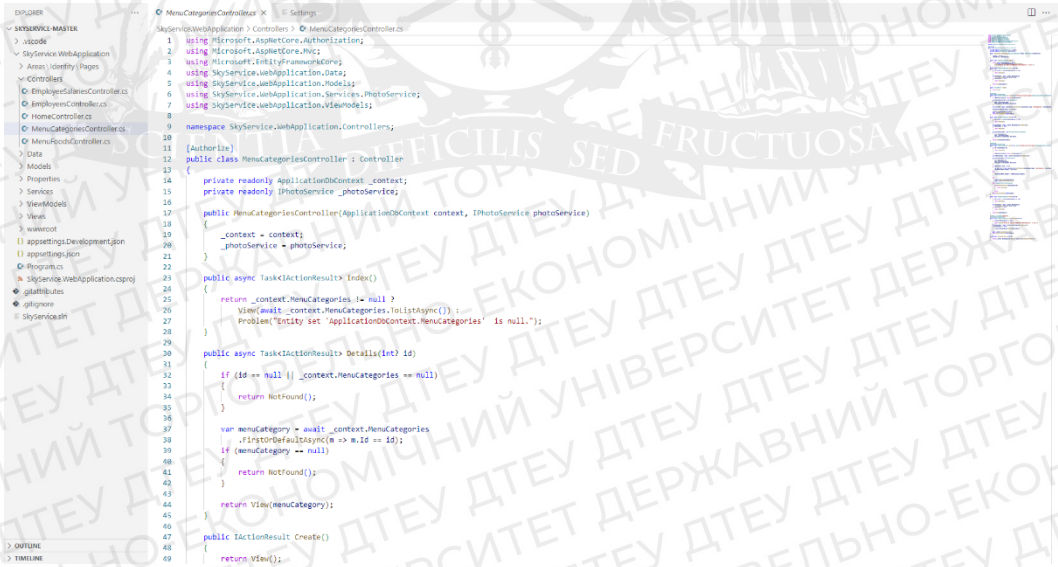

Рис. 3.1. Програма дозволяє здійснювати продажі товарів і послуг

Обробка продажів в програмній компоненті POS-рішення "SkyService" для сільськогосподарського підприємства включає такі деталі:

– сканування або введення товарів. Користувач може використовувати сканер штрих-кодів для швидкого і точного введення товарів до системи. Також є можливість вручне ввести коди товарів або використовувати інтерфейс для пошуку товарів за їх назвою або атрибутами.

– розрахунок суми покупки. Після вибору товарів програмна компонента автоматично обчислює загальну суму покупки на основі цін і кількості товарів. Вона може враховувати знижки, акції або податки, які застосовуються до окремих товарів або загальної суми.

– прийом платежів. "SkyService" підтримує різні методи оплати, включаючи готівку, кредитні картки, дебетові картки та електронні платіжні системи. Користувач може обрати відповідний метод оплати залежно від вимог клієнта.

– видача здачі. Після отримання платежу програмна компонента розраховує необхідну здачу, яку треба повернути клієнту. Вона автоматично враховує отриману суму і загальну вартість покупки, допомагаючи уникнути помилок при розрахунках.

Загальна мета цих функцій - забезпечити швидку та ефективну обробку продажів на сільськогосподарському підприємстві, дозволяючи точно визначити суму покупки, прийняти платежі різними способами і забезпечити видачу здачі клієнту.

"SkyService" дозволяє керувати товарним асортиментом підприємства. Користувач може додавати нові товари, видаляти чи змінювати існуючі записи про товари, встановлювати ціни, відстежувати кількість наявних одиниць товару і проводити інвентаризацію.

Управління товарами в програмній компоненті POS-рішення "SkyService" для сільськогосподарського підприємства включає такі деталі (див. рис. 3.2.) :

– додавання нових товарів. Користувач має можливість створювати нові записи про товари в системі. Він може ввести інформацію про назву товару, опис, атрибути (наприклад, вагу, розмір, колір тощо), ціну та інші характеристики.

– видалення та зміна існуючих записів про товари. Користувач може видаляти товари, які більше не продаються або не актуальні. Також він може змінювати існуючі записи, наприклад, змінювати ціни, оновлювати опис або змінювати атрибути товару.

– встановлення цін. Користувач може встановлювати ціни на товари. "SkyService" дозволяє встановлювати різні типи цін, такі як роздрібна ціна, оптова ціна або спеціальні знижки для певних категорій клієнтів.

– відстежування кількості наявних одиниць товару. Система дозволяє вести облік складу і відстежувати кількість наявних одиниць кожного товару. Користувач може переглядати і оновлювати цю інформацію, щоб вчасно замовляти нові запаси або виконувати виробничі замовлення.

– інвентаризація. "SkyService" надає можливість проводити інвентаризацію, що дозволяє перевіряти фактичну кількість товару на складі та порівнювати її зі значеннями в системі. Це допомагає виявляти розбіжності і виключати помилки у складському обліку.

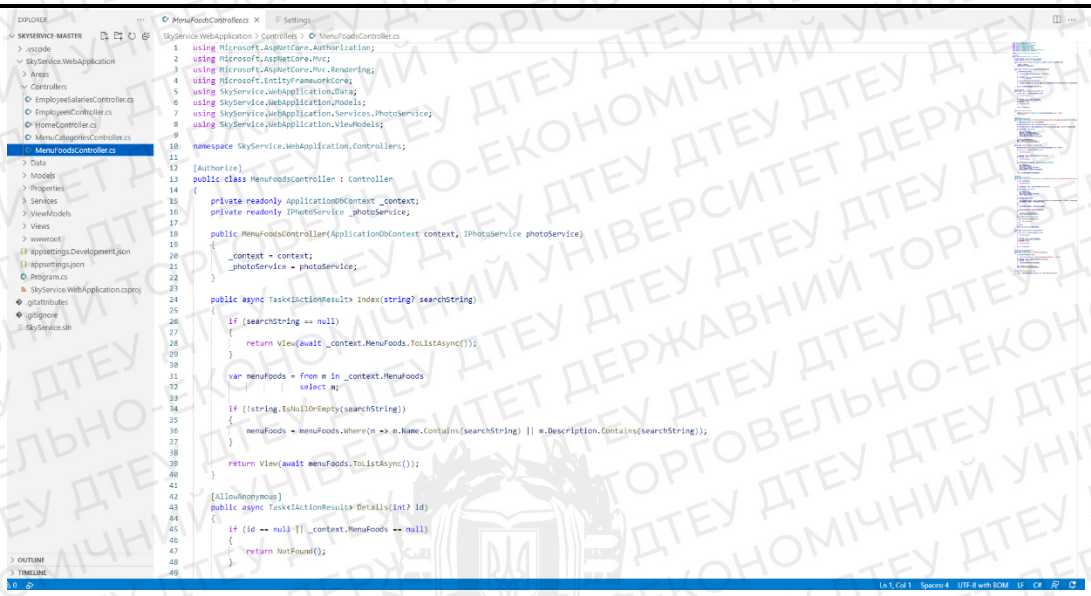

Рис. 3.2. Управління товарами в програмній компоненті POS-рішення "SkyService"

Загальна мета цих функцій - забезпечити ефективне управління товарами на сільськогосподарському підприємстві, дозволяючи додавати, змінювати і видаляти записи про товари, встановлювати ціни, відстежувати наявність товарів на складі і проводити інвентаризацію для точного обліку запасів.

POS-рішення дозволяє зберігати дані про клієнтів, включаючи контактну інформацію і історію покупок. Користувач може створювати нові профілі клієнтів, знаходити і редагувати існуючі записи та надавати знижки або бонуси клієнтам.

Керування клієнтами в програмній компоненті POS-рішення наведене на рис. 3.3.

Рис. 3.3. Керування клієнтами в програмній компоненті POS-рішення "SkyService"

Система дозволяє зберігати і оновлювати дані про клієнтів, такі як ім'я, контактна інформація (адреса, телефон, електронна пошта), дата народження та інші релевантні дані. Це дозволяє підприємству підтримувати актуальну базу даних клієнтів.

Користувач може створювати нові профілі клієнтів в системі. Він може ввести необхідну інформацію про клієнта, яка допоможе в його ідентифікації та зв'язку з ним в майбутньому.

Користувач може знаходити і редагувати існуючі записи про клієнтів для оновлення інформації або виправлення помилок. Він може змінювати контактні дані, додавати або видаляти додаткову інформацію про клієнта.

Система зберігає історію покупок кожного клієнта, включаючи деталі про придбані товари, дату покупки і вартість. Це дозволяє аналізувати покупки клієнта, розуміти його вподобання і звички споживання, а також надавати персоналізовані пропозиції та знижки.

"SkyService" надає можливість надавати знижки, бонуси або програми лояльності клієнтам. Користувач може встановлювати різні види знижок (відсоток від суми покупки, фіксовану знижку тощо) і налаштовувати бонусні програми для стимулювання повторних покупок та залучення нових клієнтів.

Загальна мета цих функцій - забезпечити зручне управління клієнтами, зберігання і оновлення їх даних, аналіз покупок і надання персоналізованих послуг для покращення взаємодії з клієнтами та збільшення їх задоволеності.

Програмна компонента "SkyService" надає можливість генерувати різноманітні звіти, що стосуються продажів, складу товарів, прибутку, витрат та інших аспектів діяльності підприємства. Користувач може аналізувати ці дані для прийняття управлінських рішень.

Звітність в програмній компоненті POS-рішення "SkyService" для сільськогосподарського підприємства включає такі деталі:

1. Генерація звітів: Користувач може генерувати різноманітні звіти, які стосуються різних аспектів діяльності підприємства. Ці звіти можуть включати звіти про продажі, склад товарів, прибуток та витрати, фінансові звіти, звіти про рух товарів тощо.

2. Звіти про продажі: Система може генерувати звіти, які надають детальну інформацію про продажі, такі як загальна сума продажів, кількість проданих одиниць товарів, найпопулярніші товари або послуги, структура продажів за період часу тощо. Це дозволяє аналізувати тенденції та ефективність продажів.

3. Звіти про склад товарів: Система може генерувати звіти, які відображають стан складу товарів, включаючи кількість наявних одиниць кожного товару, загальну вартість запасів, рух товарів (прихід, відвантаження, повернення тощо) і іншу інформацію, яка допомагає контролювати запаси.

4. Фінансові звіти: Система може генерувати фінансові звіти, такі як звіт про прибуток і збитки, звіт про грошові потоки, баланс підприємства тощо. Ці

звіти надають користувачеві уявлення про фінансовий стан підприємства і його рентабельність.

5. Аналітичні звіти: "SkyService" дозволяє генерувати аналітичні звіти, які детально аналізують дані підприємства. Це можуть бути звіти про витрати, рентабельність товарів, ефективність маркетингових акцій тощо. Ці звіти допомагають зрозуміти ефективність бізнесу і приймати управлінські рішення.

Загальна мета звітності - надати користувачу необхідну інформацію для аналізу діяльності підприємства, прийняття управлінських рішень, виявлення потенційних проблем або можливостей та вдосконалення бізнес-процесів.

"SkyService" може інтегруватися з обліковою системою підприємства для автоматичного оновлення даних про продажі, запаси та іншу інформацію. Це дозволяє забезпечити точність і уніфікованість даних у всіх відділах підприємства.

Інтеграція з обліковою системою в програмній компоненті POS-рішення "SkyService" для сільськогосподарського підприємства включає наступні деталі:

1. Автоматичне оновлення даних: "SkyService" може підключатися до облікової системи підприємства, що використовується для фінансового та бухгалтерського обліку. Ця інтеграція дозволяє автоматично оновлювати дані про продажі, витрати, прибуток та інші фінансові показники в обох системах без необхідності вручного введення даних.

2. Уніфікованість даних: Інтеграція з обліковою системою допомагає забезпечити уніфікованість даних у всіх відділах підприємства. Це означає, що дані про продажі, запаси, витрати та іншу інформацію будуть однакові і актуальні як у POS-системі, так і в обліковій системі. Це спрощує ведення обліку, уникнення помилок та забезпечує точність фінансових даних.

3. Синхронізація запасів: Інтеграція з обліковою системою дозволяє автоматично оновлювати дані про запаси товарів. Коли відбуваються продажі або поповнення запасів через POS-систему, ця інформація передається в облікову систему, що дозволяє забезпечити точний облік запасів та уникнути ситуацій перепродажу або вичерпання товарів.

4. Фінансовий облік: Інтеграція дозволяє автоматично передавати фінансові дані з POS-системи до облікової системи. Це включає дані про продажі, оплати, податки, знижки та інші фінансові транзакції. Така автоматизація спрощує процеси обліку та звітності, зменшує ймовірність помилок та забезпечує точність фінансової інформації.

Загальна мета інтеграції з обліковою системою - забезпечити автоматизацію обліку та обмін даними між POS-системою та обліковою системою, що допомагає підприємству підтримувати точність, уніфікованість та актуальність даних у всіх відділах та процесах діяльності.

POS-рішення "SkyService" підтримує різні типи платежів, такі як готівка, кредитні картки, електронні платежі та інші. Користувач може обирати зручний спосіб оплати для клієнтів. Система дозволяє обробляти платежі готівкою. Користувач може вводити суму покупки і приймати оплату в готівці. Після цього система розраховує здачу і друкує чек для клієнта. "SkyService" підтримує оплату за допомогою кредитних карток. Користувач може використовувати термінал для зчитування і обробки інформації з кредитних карток, що дозволяє здійснювати безпечні та швидкі транзакції. POS-система також підтримує різні електронні платіжні системи, такі як платіжні шлюзи, мобільні платіжні додатки, електронні гаманці тощо. Користувач може приймати платежі через ці системи, що забезпечує зручність для клієнтів, які використовують ці електронні методи оплати. "SkyService" може підтримувати й інші типи платежів, такі як чеки, перекази, корпоративні карти

тощо. Залежно від потреб підприємства, користувач може налаштувати систему для обробки цих типів платежів.

Програмна компонента "SkyService" розроблена з урахуванням модульної структури, що дозволяє розширювати її функціональність за потребами підприємства. Додаткові модулі можуть бути підключені для виконання специфічних завдань, таких як управління складським обліком чи забезпечення програми лояльності.

3.2. Особливості впровадження розробленої програмної компоненти

Впровадження розробленої програмної компоненти "SkyService" на підприємство включає деякі особливості, які необхідно врахувати. Перед впровадженням "SkyService" варто провести аналіз потреб підприємства і зрозуміти, які функціональності програмної компоненти будуть найбільш корисні та важливі для підприємства. Це допоможе налаштувати систему відповідно до потреб бізнесу.

Впровадження "SkyService" вимагає налаштування системи відповідно до специфічних потреб підприємства. Це включає налаштування товарного асортименту, цін, податків, знижок та інших параметрів. Крім того, необхідно забезпечити інтеграцію з існуючими системами, такими як облікова система, щоб забезпечити обмін даними.

Необхідно налаштувати компоненту "SkyService" з урахуванням асортименту товарів, які продаються на підприємстві. Це включає додавання нових товарів та встановлення їхніх атрибутів, таких як назва, код, опис, ціна та інші характеристики.

Система "SkyService" повинна бути налаштована для встановлення цін на товари відповідно до внутрішніх правил підприємства. Це може включати

встановлення постійних цін, знижок або акційних пропозицій для певних товарів або категорій товарів.

Враховуючи податкові вимоги та законодавство, "SkyService" повинна бути налаштована для обліку та автоматичного розрахунку податків на товари та послуги, які продаються. Це може включати різні види податків, такі як ПДВ або акцизний збір, і встановлення їхніх ставок.

Якщо на підприємстві передбачається використання системи знижок для клієнтів, "SkyService" повинна бути налаштована для встановлення різних видів знижок та їхніх умов. Це можуть бути постійні знижки, знижки на обсяг або спеціальні пропозиції для певних груп клієнтів.

Для забезпечення обміну даними і спільної роботи з існуючими системами на підприємстві, "SkyService" повинна бути налаштована для інтеграції з обліковою системою, системою управління запасами або іншими системами, що використовуються. Це дозволяє автоматично оновлювати дані про продажі, запаси та іншу інформацію.

При цьому функціонування самої програмної компоненти відбувається на основі класів (див. рис. 3.4.):

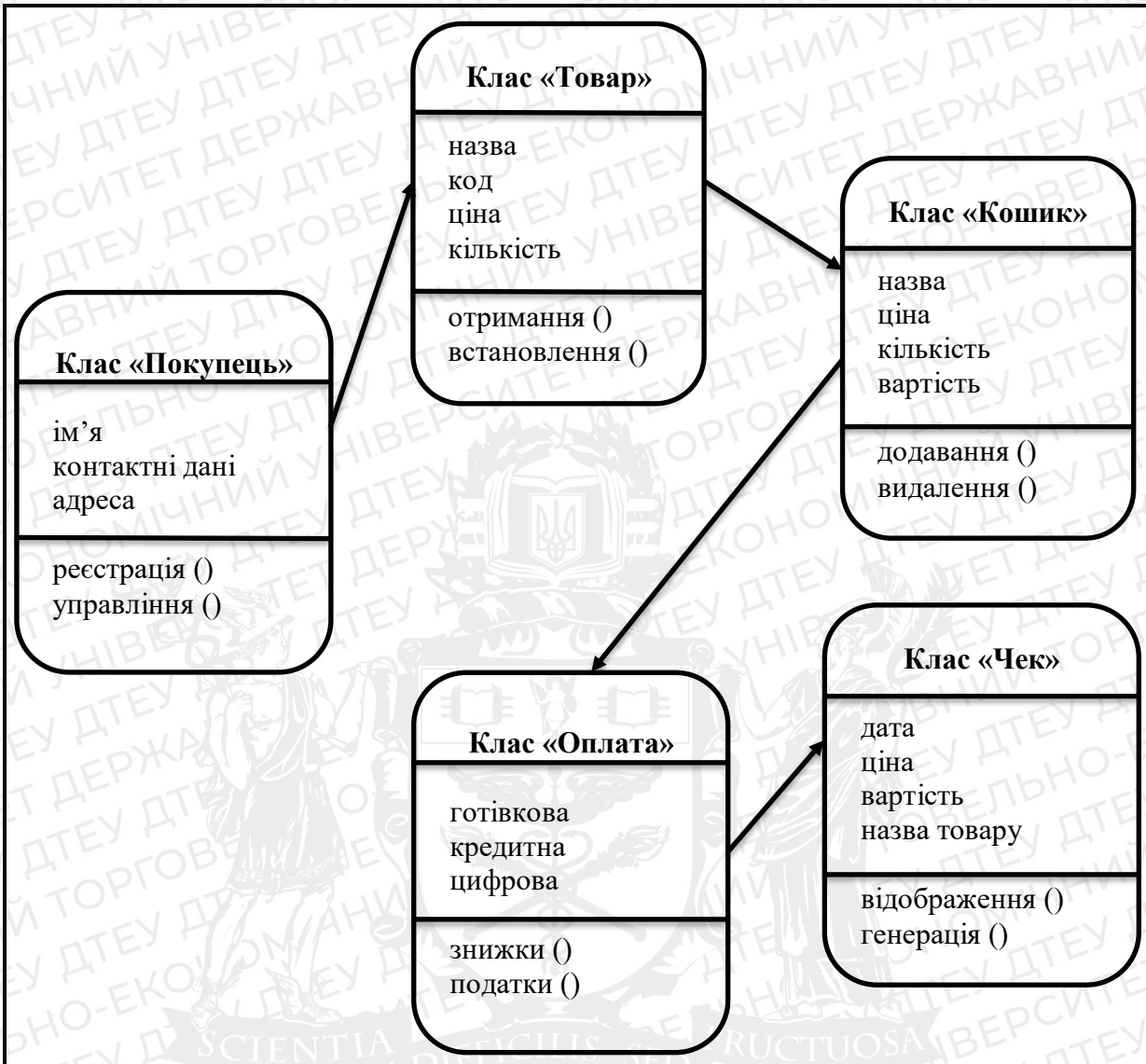

Рис. 3.4. Діаграма класів програмної компоненти "SkyService"

Так, програма пропонує 5 класів. Розглянемо детальніще кожен з них.

Клас «Покупець» необхідний для збереження даниих про користувачів системи та збереження їх контактних даних. Це необхідно для планування маркетингової діяльності підприємства та власне зв'язку з клієнтом у разі здійснення ним покупки.

Клас «Товар» нгеобхідний для ідентифікації продукції підприємства. Обов'язково має бути код товару, адже товари можуть мати однакову назву,

однак при цьому відрізнятись за ціною, брендом тощо. Тому в даному класі ключем виступає код.

Клас «Кошик» необхідний для переміщення туди товарів, які сподобалися покупцю і які він має намір придбати. Покупець обирає товар та відправляє його у кошик, де буде зазначена назва даного товару, його ціна, кількість та вартість. Крім того, покупець матиме змогу видалити товар з кошика в разі зміни своїх уподобань.

Клас «Оплата» необхідний для проведення оплати за товар, який був у кошику. Клієнт може розрахуватися готівкою, банківською картою або іншими інструментами (електронні гроші, криптовалюта тощо). Крім того, мають бути враховані наявні знижки та податки, які підприємство сплачуватиме з отриманого доходу.

Клас «Чек» необхідний для збереження інформацію про покупку. Так, він містить у собі назви товарів, дату придбання, вартість та ціну.

Уважне налаштування та інтеграція програмної компоненти "SkyService" допоможе забезпечити її ефективне функціонування на підприємстві, враховуючи специфічні потреби та вимоги бізнесу.

Для забезпечення безпеки даних і обмеження доступу до різних функціональностей "SkyService" варто належним чином налаштувати права доступу користувачів. Необхідно визначити, які працівники мають доступ до різних функцій системи і які обмеження повинні бути встановлені.

Впровадження нової програмної компоненти вимагає навчання персоналу. Користувачі повинні бути ознайомлені з функціональністю "SkyService" і навчитися використовувати його ефективно. Крім того, необхідно забезпечити підтримку та технічну допомогу для розробленої програмної компоненти, щоб вирішувати потенційні проблеми та відповідати на запити користувачів.

Рекомендується провести тестування "SkyService" перед повним впровадженням на підприємстві. Тестування допоможе виявити та виправити помилки, а також переконатися, що програмна компонента працює належним чином перед її використанням у повсякденній діяльності підприємства. Рекомендується впроваджувати систему поетапно, починаючи з одного відділу або магазину, щоб забезпечити плавний перехід і виправлення можливих проблем.

Врахування цих особливостей під час впровадження розробленої програмної компоненти "SkyService" допоможе забезпечити успішну і ефективну роботу системи на сільськогосподарському підприємстві.

3.3. Висновок до розділу 3

POS-рішення "SkyService" для сільськогосподарського підприємства є комплексною програмною компонентою, яка надає широкий спектр функціональностей для керування продажами та іншими аспектами діяльності підприємства. Обробка продажів дозволяє здійснювати продажі товарів і послуг, сканувати або вводити товари, розраховувати суму покупки, приймати платежі та видачу здачі. Управління товарами дозволяє додавати, видаляти та змінювати записи про товари, встановлювати ціни, відстежувати кількість наявних одиниць товару та проводити інвентаризацію. Керування клієнтами дозволяє зберігати дані про клієнтів, створювати профілі, редагувати існуючі записи та надавати знижки або бонуси клієнтам. Звітність забезпечує генерацію різноманітних звітів щодо продажів, складу товарів, прибутку, витрат та інших аспектів діяльності підприємства, що допомагає управлінцям приймати обґрунтовані рішення. Інтеграція з обліковою системою дозволяє автоматичне оновлення даних про продажі, запаси та іншу інформацію, що сприяє точності та уніфікованості даних у всіх відділах підприємства. Підтримка різних типів платежів дозволяє клієнтам

використовувати різні способи оплати, такі як готівка, кредитні картки, електронні платежі тощо, що забезпечує зручність і гнучкість.

ВИСНОВОК ТА ПРОПОЗИЦІЇ

POS-рішення в сільськогосподарському секторі мають свої особливості, які враховують специфіку галузі, зокрема зважування продукції і обробка касових операцій для різних категорій товарів. Важливо, щоб POS-системи в сільському господарстві могли керувати запасами сільськогосподарської продукції, включаючи постачання і перевірку наявності товарів на складі. Інтеграція з іншими сільськогосподарськими системами дозволяє автоматизувати обмін даними і підвищує точність та ефективність операцій. Підтримка мобільних пристроїв дозволяє працівникам проводити операції на місці, що особливо важливо для розподіленої робочої сили в сільському господарстві. POS-системи повинні надавати зручні засоби для створення звітів про продажі, інвентаризацію та аналіз продуктивності, що допомагає управляти бізнесом.

Ці особливості допомагають сільськогосподарським підприємствам підвищити ефективність операцій, забезпечити точність обліку і управляти своїм бізнесом з використанням POS-рішень. POS-рішення призначене для забезпечення ефективного управління продажами та обліку в сільськогосподарському підприємстві. Основною метою створення POS-рішення є поліпшення процесу продажів, оптимізація обліку товарів, автоматизація фінансових операцій та забезпечення точності та надійності обробки даних.

Система повинна забезпечувати ефективний і безперебійний інформаційний обмін між компонентами системи, режими функціонування, діагностику та управління системою. Система повинна мати високу ступінь відповідності своєму призначенню та бути гнучкою для змін в

сільськогосподарському середовищі, повинна бути стійкою до різних умов експлуатації та забезпечувати надійність технічних засобів і програмного забезпечення.

Система повинна відповідати принципам ергономіки та технічної естетики для забезпечення зручного та приємного користувацького досвіду, мати вимоги до експлуатації, технічного обслуговування, ремонту і зберігання компонентів системи, щоб забезпечити їхню тривалу та надійну роботу.

Java, C#, Python і JavaScript є популярними мовами для розробки POS-рішень. IDE допомагають створювати, налагоджувати і тестувати код. Приклади популярних IDE включають Eclipse, Visual Studio, PyCharm і Visual Studio Code. Використання фреймворків спрощує розробку і забезпечує швидку реалізацію функціональності. Наприклад, Spring, Hibernate, ASP.NET, Entity Framework, Django і Flask є популярними фреймворками для розробки POS-рішень.

Для зберігання даних POS-рішення можна використовувати реляційні бази даних, такі як MySQL, PostgreSQL або Microsoft SQL Server. Нереляційні бази даних, такі як MongoDB або Redis, також можуть бути використані. Для розробки веб-орієнтованої компоненти POS-рішення використовуються HTML, CSS і JavaScript. Фреймворки, такі як Angular, React і Vue.js, допомагають створювати потужні користувацькі інтерфейси.

Взаємодія з різними аспектами POS-системи вимагає використання API і бібліотек. Платіжні шлюзи, такі як Stripe або PayPal, можуть бути використані для обробки платежів. Інструменти тестування, такі як JUnit, NUnit і pytest, допомагають забезпечити якість програмної компоненти POS-рішення.

POS-рішення "SkyService" для сільськогосподарського підприємства є комплексною програмною компонентою, яка надає широкий спектр функціональностей для керування продажами та іншими аспектами

діяльності підприємства. Обробка продажів дозволяє здійснювати продажі товарів і послуг, сканувати або вводити товари, розраховувати суму покупки, приймати платежі та видачу здачі. Управління товарами дозволяє додавати, видаляти та змінювати записи про товари, встановлювати ціни, відстежувати кількість наявних одиниць товару та проводити інвентаризацію. Керування клієнтами дозволяє зберігати дані про клієнтів, створювати профілі, редагувати існуючі записи та надавати знижки або бонуси клієнтам. Звітність забезпечує генерацію різноманітних звітів щодо продажів, складу товарів, прибутку, витрат та інших аспектів діяльності підприємства, що допомагає управлінцям приймати обґрунтовані рішення. Інтеграція з обліковою системою дозволяє автоматичне оновлення даних про продажі, запаси та іншу інформацію, що сприяє точності та уніфікованості даних у всіх відділах підприємства. Підтримка різних типів платежів дозволяє клієнтам використовувати різні способи оплати, такі як готівка, кредитні картки, електронні платежі тощо, що забезпечує зручність і гнучкість.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ari Lerner, Felipe Coury, and Nate Murray. Ng-Book 2: The Complete Book on Angular 2. 2016 – 609 ст.

2. Dan Vanderkam. Effective TypeScript: 62 Specific Ways to Improve Your TypeScript. 2019 – 266 ст.

3. David Herman. Effective JavaScript: 68 Specific Ways to Harness the Power of JavaScript. 2012 – 240 ст.

4. Electronic Prescription Service (EPS). URL

5. Fuscaldo D. How to Choose a POS Cash Register. *Business.* 2023. URL: https://www.business.com/articles/pos-cash-register/

6. Jennifer Niederst Robbins. Learning Web Design. 2012 – 808 ст.

7. Jeremy Wilken. Angular in Action. 2018 – 320 ст.

8. Jon Duckett. HTML & CSS: Design and Build Web Sites. 2011 – 514 ст.

9. Mario Casciaro. Node.js Design Patterns. 2014 – 520 ст.

10. Model-View-Controller (MVC) – Режим доступу: https://medium.com/datadriveninvestor/model-view-controller-mvc75bcb0103d66

11. Sorensen E. Cash register vs. POS system – what's the difference? *Mobile transaction.* 2022. URL: https://www.mobiletransaction.org/cash-register-vs-pos-system/

12. Tricks traders use to evade billions of francs in taxes. *The New Times.* 2014. URL: https://www.newtimes.co.rw/article/112177/News/tricks-traders-use-to-evade-billions-of-francs-in-taxes

					ДТЕУ 121 06-17.БР			
Зм.	*Аркуш*	*№ докум.*	*Підпис*	*Дата*				
Зав. каф.		Криворучко О.В.		23.12.22	*Програмна компонента POS-рішення «Skyservise» для підприємства сільськогосподарського господарства*	*Стадія*	*Аркуш*	*Аркушів*
Керівник		Хорольська К.В.		23.12.22		*СВД*	*58*	*80*
Гарант		Рзаєва С.Л.		23.12.22		Факультет інформаційних технологій *4 курс, 6 група*		
Розробив		Максимчук Р.В.		23.12.22	*Список використаних джерел*			

13.Клієнт-серверна архітектура та ролі серверів. – Режим доступу: https://medium.com/@IvanZmerzlyi/клієнт-серверна-архітектура-та-ролісерверів-9893d8048229

ДОДАТКИ

ДОДАТОК А

Код програмної компоненти

```
@page
@model LoginModel

@{
    ViewData["Title"] = "Log in";
}

<h1>@ViewData["Title"]</h1>
<div class="row">
    <div class="col-md-4">
        <section>
            <form id="account" method="post">
                <h2>Use an account to log in.</h2>
                <hr />
                <div asp-validation-summary="ModelOnly" class="text-danger"
role="alert"></div>
                <div class="form-floating mb-3">
                    <input asp-for="Input.Email" class="form-control"
autocomplete="username" aria-required="true" placeholder="name@example.com" />
                    <label asp-for="Input.Email" class="form-label">Email</label>
                    <span asp-validation-for="Input.Email" class="text-
danger"></span>
                </div>
                <div class="form-floating mb-3">
                    <input asp-for="Input.Password" class="form-control"
autocomplete="current-password" aria-required="true" placeholder="password" />
                    <label asp-for="Input.Password" class="form-
label">Password</label>
                    <span asp-validation-for="Input.Password" class="text-
danger"></span>
                </div>
                <div class="checkbox mb-3">
                    <label asp-for="Input.RememberMe" class="form-label">
                        <input class="form-check-input" asp-for="Input.RememberMe" />
                        @Html.DisplayNameFor(m => m.Input.RememberMe)
                    </label>
                </div>
                <div>
                    <button id="login-submit" type="submit" class="w-100 btn btn-lg
btn-primary">Log in</button>
                </div>
            </form>
        </section>
    </div>
</div>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}

@page
@model RegisterModel
@{
    ViewData["Title"] = "Register";
}
```

```
<h1>@ViewData["Title"]</h1>

<div class="row">
    <div class="col-md-4">
        <form id="registerForm" asp-route-returnUrl="@Model.ReturnUrl" method="post">
            <h2>Create a new account.</h2>
            <hr />
            <div asp-validation-summary="ModelOnly" class="text-danger"
role="alert"></div>
            <div class="form-floating mb-3">
                <input asp-for="Input.Email" class="form-control"
autocomplete="username" aria-required="true" placeholder="name@example.com" />
                <label asp-for="Input.Email">Email</label>
                <span asp-validation-for="Input.Email" class="text-danger"></span>
            </div>
            <div class="form-floating mb-3">
                <input asp-for="Input.Password" class="form-control"
autocomplete="new-password" aria-required="true" placeholder="password" />
                <label asp-for="Input.Password">Password</label>
                <span asp-validation-for="Input.Password" class="text-danger"></span>
            </div>
            <div class="form-floating mb-3">
                <input asp-for="Input.ConfirmPassword" class="form-control"
autocomplete="new-password" aria-required="true" placeholder="password" />
                <label asp-for="Input.ConfirmPassword">Confirm Password</label>
                <span asp-validation-for="Input.ConfirmPassword" class="text-
danger"></span>
            </div>
            <button id="registerSubmit" type="submit" class="w-100 btn btn-lg btn-
primary">Register</button>
        </form>
    </div>
    <div class="col-md-6 col-md-offset-2">
        <section>
            <h3>Use another service to register.</h3>
            <hr />
            @{
                if ((Model.ExternalLogins?.Count ?? 0) == 0)
                {
                    <div>
                        <p>
                            There are no external authentication services configured.
See this <a href="https://go.microsoft.com/fwlink/?LinkID=532715">article
                            about setting up this ASP.NET application to support
logging in via external services</a>.
                        </p>
                    </div>
                }
                else
                {
                    <form id="external-account" asp-page="./ExternalLogin" asp-route-
returnUrl="@Model.ReturnUrl" method="post" class="form-horizontal">
                        <div>
                            <p>
                                @foreach (var provider in Model.ExternalLogins!)
                                {
                                    <button type="submit" class="btn btn-primary"
name="provider" value="@provider.Name" title="Log in using your @provider.DisplayName
account">@provider.DisplayName</button>
                                }
                            </p>
```

```
                    </div>
                </form>
            }
        }
    </section>
</div>
</div>

@section Scripts {
    <partial name="_ValidationScriptsPartial" />
}
<environment include="Development">
    <script src="~/Identity/lib/jquery-validation/dist/jquery.validate.js"></script>
    <script src="~/Identity/lib/jquery-validation-
unobtrusive/jquery.validate.unobtrusive.js"></script>
</environment>
<environment exclude="Development">
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-
validate/1.17.0/jquery.validate.min.js"
            asp-fallback-src="~/Identity/lib/jquery-
validation/dist/jquery.validate.min.js"
            asp-fallback-test="window.jQuery && window.jQuery.validator"
            crossorigin="anonymous"
            integrity="sha384-
rZfj/ogBloos6wzLGpPkkOr/gpkBNLZ6b6yLy4o+ok+t/SAKlL5mvXLr0OXNi1Hp">
    </script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validation-
unobtrusive/3.2.11/jquery.validate.unobtrusive.min.js"
            asp-fallback-src="~/Identity/lib/jquery-validation-
unobtrusive/jquery.validate.unobtrusive.min.js"
            asp-fallback-test="window.jQuery && window.jQuery.validator &&
window.jQuery.validator.unobtrusive"
            crossorigin="anonymous"
            integrity="sha384-
R3vNCHsZ+A2Lo3d5A6XNP7fdQkeswQWTIPfiYwSpEP3YV079R+93YzTeZRah7f/F">
    </script>
</environment>

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using SkyService.WebApplication.Data;
using SkyService.WebApplication.Models;
using SkyService.WebApplication.ViewModels;

namespace SkyService.WebApplication.Controllers;

[Authorize]
public class EmployeeSalariesController : Controller
{
    private readonly ApplicationDbContext _context;

    public EmployeeSalariesController(ApplicationDbContext context)
    {
        _context = context;
    }

    public async Task<IActionResult> Index(int id)
    {
        var employee = await _context.Employees.FirstOrDefaultAsync(e => e.Id == id);
```

```
        var salaries = await _context.Salary.Where(e => e.EmployeeId ==
id).ToListAsync();

        return _context.Salary.Any(e => e.EmployeeId == id) ?
View(salaries.Select(salary => new EmployeeSalaryViewModel
        {
            Id = salary.Id,
            EmployeeId = salary.EmployeeId,
            Date = salary.Date,
            Salary = salary.Salary,
            Hours = salary.Hours,
            VacationDays = salary.VacationDays,
            SickDays = salary.SickDays,
            Overtime = salary.Overtime,
            Bonus = salary.Bonus,
            Deductions = salary.Deductions,
            Comment = salary.Comment,
            EmployeeName = employee?.Name,
            Total = salary.Salary + salary.Bonus - salary.Deductions
        })) : RedirectToAction("Create", new { id });
    }

    public async Task<IActionResult> TotalSalary()
    {
        var salaries = await _context.Salary.ToListAsync();
        var salariesViewModel = salaries.Select(salary => new EmployeeSalaryViewModel
            {
                Id = salary.Id,
                EmployeeId = salary.EmployeeId,
                Date = salary.Date,
                Salary = salary.Salary,
                Hours = salary.Hours,
                VacationDays = salary.VacationDays,
                SickDays = salary.SickDays,
                Overtime = salary.Overtime,
                Bonus = salary.Bonus,
                Deductions = salary.Deductions,
                Comment = salary.Comment,
                EmployeeName = _context.Employees.FirstOrDefault(e => e.Id ==
salary.EmployeeId)?.Name,
                Total = salary.Salary + salary.Bonus - salary.Deductions
        })
            .ToList();

        var salaryViewModelDetailed = new SalaryViewModel
        {
            Salaries = salariesViewModel,
            Salary = await _context.Salary.SumAsync(e => e.Salary),
            Hours = await _context.Salary.SumAsync(e => e.Hours),
            VacationDays = await _context.Salary.SumAsync(e => e.VacationDays),
            SickDays = await _context.Salary.SumAsync(e => e.SickDays),
            Overtime = await _context.Salary.SumAsync(e => e.Overtime),
            Bonus = await _context.Salary.SumAsync(e => e.Bonus),
            Deductions = await _context.Salary.SumAsync(e => e.Deductions),
            Total = await _context.Salary.SumAsync(e => e.Salary + e.Bonus -
e.Deductions)
        };

        return View(salaryViewModelDetailed);
    }

    public IActionResult Create(int id)
```

```
    {
        var employeeSalary = new EmployeeSalary
        {
            EmployeeId = id,
            Date = DateTime.Now
        };

        return View(employeeSalary);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
Create([Bind("EmployeeId,Date,Salary,Hours,VacationDays,SickDays,Overtime,Bonus,Deduc
tions,Comment")] EmployeeSalary employeeSalary)
    {
        if (!ModelState.IsValid) return View(employeeSalary);

        _context.Add(employeeSalary);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index), new { id = employeeSalary.EmployeeId
});
    }

    public async Task<IActionResult> Edit(int? id)
    {
        if (id == null || _context.Salary == null)
        {
            return NotFound();
        }

        var employeeSalary = await _context.Salary.FindAsync(id);
        if (employeeSalary == null)
        {
            return NotFound();
        }
        return View(employeeSalary);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(int id,
[Bind("Id,EmployeeId,Date,Salary,Hours,VacationDays,SickDays,Overtime,Bonus,Deduction
s,Comment")] EmployeeSalary employeeSalary)
    {
        if (id != employeeSalary.Id)
        {
            return NotFound();
        }

        if (ModelState.IsValid)
        {
            try
            {
                _context.Update(employeeSalary);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!EmployeeSalaryExists(employeeSalary.Id))
                {
                    return NotFound();
```

```
                }
                else
                {
                    throw;
                }
            }
            return RedirectToAction(nameof(Index), new { id =
employeeSalary.EmployeeId });
        }
        return View(employeeSalary);
    }

    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null || _context.Salary == null)
        {
            return NotFound();
        }

        var employeeSalary = await _context.Salary
            .FirstOrDefaultAsync(m => m.Id == id);
        if (employeeSalary == null)
        {
            return NotFound();
        }

        return View(employeeSalary);
    }

    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        if (_context.Salary == null)
        {
            return Problem("Entity set 'ApplicationDbContext.Salary'  is null.");
        }
        var employeeSalary = await _context.Salary.FindAsync(id);
        if (employeeSalary != null)
        {
            _context.Salary.Remove(employeeSalary);
        }

        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }

    private bool EmployeeSalaryExists(int id)
    {
        return (_context.Salary?.Any(e => e.Id == id)).GetValueOrDefault();
    }
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using SkyService.WebApplication.Data;
using SkyService.WebApplication.Models;
using SkyService.WebApplication.Services.PhotoService;
using SkyService.WebApplication.ViewModels;

namespace SkyService.WebApplication.Controllers;

[Authorize]
```

```
public class EmployeesController : Controller
{
    private readonly ApplicationDbContext _context;
    private readonly IPhotoService _photoService;

    public EmployeesController(ApplicationDbContext context, IPhotoService
photoService)
    {
        _context = context;
        _photoService = photoService;
    }

    public async Task<IActionResult> Index(string? searchString)
    {
        if (searchString == null)
        {
            return View(await _context.Employees.ToListAsync());
        }

        //get employees by search string
        var employees = from e in _context.Employees
                        select e;

        if (!string.IsNullOrEmpty(searchString))
        {
            employees = employees.Where(e => e.Name.Contains(searchString));
        }

        return View(await employees.ToListAsync());
    }

    public async Task<IActionResult> Details(int? id)
    {
        if (id == null || _context.Employees == null)
        {
            return NotFound();
        }

        var employee = await _context.Employees
            .FirstOrDefaultAsync(m => m.Id == id);
        if (employee == null)
        {
            return NotFound();
        }

        return View(employee);
    }

    public IActionResult Create()
    {
        return View();
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
Create([Bind("Id,Name,ProfilePicture,Phone,Address,City,Position,SalaryPerHour,HoursP
erWeek,VacationDays,SickDays,Deductions")] EmployeeOperationViewModel employee)
    {
        if (!ModelState.IsValid) return View(employee);
```

```
            var profilePictureUrl = await
        _photoService.SavePhotoAsync(employee.ProfilePicture, "employees", employee.Name + "
        " + employee.Phone + "_profile_picture");

            var employeeModel = new Employee
            {
                Name = employee.Name,
                ProfilePictureUrl = profilePictureUrl,
                Phone = employee.Phone,
                Address = employee.Address,
                City = employee.City,
                Position = employee.Position
            };

            _context.Add(employeeModel);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null || _context.Employees == null)
            {
                return NotFound();
            }

            var employee = await _context.Employees.FindAsync(id);
            if (employee == null)
            {
                return NotFound();
            }

            var employeeViewModel = new EmployeeOperationViewModel
            {
                Id = employee.Id,
                Name = employee.Name,
                Phone = employee.Phone,
                Address = employee.Address,
                City = employee.City,
                Position = employee.Position
            };

            return View(employeeViewModel);
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id,
    [Bind("Id,Name,ProfilePicture,Phone,Address,City,Position,SalaryPerHour,HoursPerWeek,
    VacationDays,SickDays,Deductions")] EmployeeOperationViewModel employee)
        {
            if (id != employee.Id)
            {
                return NotFound();
            }

            if (!ModelState.IsValid) return View(employee);

            var employeeOld = await _context.Employees.FindAsync(id);

            var employeeModel = new Employee
            {
```

```
            Id = employee.Id,
            Name = employee.Name,
            Phone = employee.Phone,
            Address = employee.Address,
            City = employee.City,
            Position = employee.Position
        };

        if (employee.ProfilePicture != null)
        {
            var profilePictureUrl = await
_photoService.SavePhotoAsync(employee.ProfilePicture, "employees", employee.Name + "
" + employee.Phone + "_profile_picture");
            employeeModel.ProfilePictureUrl = profilePictureUrl;
        }
        else
        {
            employeeModel.ProfilePictureUrl = employeeOld.ProfilePictureUrl;
        }

        try
        {
            _context.Update(employeeModel);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!EmployeeExists(employee.Id))
            {
                return NotFound();
            }

            throw;
        }
        return RedirectToAction(nameof(Index));
    }

    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null || _context.Employees == null)
        {
            return NotFound();
        }

        var employee = await _context.Employees
            .FirstOrDefaultAsync(m => m.Id == id);
        if (employee == null)
        {
            return NotFound();
        }

        return View(employee);
    }

    [HttpPost, ActionName("Delete")]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        if (_context.Employees == null)
        {
            return Problem("Entity set 'ApplicationDbContext.Employees'  is null.");
        }
```

```
            var employee = await _context.Employees.FindAsync(id);
            if (employee != null)
            {
                _context.Employees.Remove(employee);
            }

            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private bool EmployeeExists(int id)
        {
            return (_context.Employees?.Any(e => e.Id == id)).GetValueOrDefault();
        }
    }
    using Microsoft.AspNetCore.Mvc;
    using SkyService.WebApplication.Models;
    using System.Diagnostics;
    using SkyService.WebApplication.Data;

    namespace SkyService.WebApplication.Controllers;

    public class HomeController : Controller
    {
        private readonly ApplicationDbContext _context;

        public HomeController(ILogger<HomeController> logger, ApplicationDbContext
    context)
        {
            _context = context;
        }

        public IActionResult Index()
        {
            var categories = _context.MenuCategories.ToList();
            var foods = _context.MenuFoods.ToList();
            categories.ForEach(c => c.MenuFoods = foods.Where(f => f.MenuCategoryId ==
    c.Id).ToList());
            return View(categories);
        }

        public IActionResult Privacy()
        {
            return View();
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore =
    true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??
    HttpContext.TraceIdentifier });
        }
    }
    using Microsoft.AspNetCore.Authorization;
    using Microsoft.AspNetCore.Mvc;
    using Microsoft.EntityFrameworkCore;
    using SkyService.WebApplication.Data;
    using SkyService.WebApplication.Models;
    using SkyService.WebApplication.Services.PhotoService;
    using SkyService.WebApplication.ViewModels;
```

```
namespace SkyService.WebApplication.Controllers;

[Authorize]
public class MenuCategoriesController : Controller
{
    private readonly ApplicationDbContext _context;
    private readonly IPhotoService _photoService;

    public MenuCategoriesController(ApplicationDbContext context, IPhotoService
photoService)
    {
        _context = context;
        _photoService = photoService;
    }

    public async Task<IActionResult> Index()
    {
        return _context.MenuCategories != null ?
            View(await _context.MenuCategories.ToListAsync()) :
            Problem("Entity set 'ApplicationDbContext.MenuCategories'  is null.");
    }

    public async Task<IActionResult> Details(int? id)
    {
        if (id == null || _context.MenuCategories == null)
        {
            return NotFound();
        }

        var menuCategory = await _context.MenuCategories
            .FirstOrDefaultAsync(m => m.Id == id);
        if (menuCategory == null)
        {
            return NotFound();
        }

        return View(menuCategory);
    }

    public IActionResult Create()
    {
        return View();
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Create([Bind("Id,Name,Description,Image")]
MenuCategoryOperationViewModel menuCategory)
    {
        if (!ModelState.IsValid) return View(menuCategory);

        var menuCategoryToCreate = new MenuCategory
        {
            Name = menuCategory.Name,
            Description = menuCategory.Description
        };

        var pictureUrl = await _photoService.SavePhotoAsync(menuCategory.Image,
"menuCategories", menuCategory.Name + "_image");

        menuCategoryToCreate.ImageUrl = pictureUrl;
```

```
            _context.Add(menuCategoryToCreate);
            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        public async Task<IActionResult> Edit(int? id)
        {
            if (id == null || _context.MenuCategories == null)
            {
                return NotFound();
            }

            var menuCategory = await _context.MenuCategories.FindAsync(id);
            if (menuCategory == null)
            {
                return NotFound();
            }

            var menuCategoryToEdit = new MenuCategoryOperationViewModel
            {
                Id = menuCategory.Id,
                Name = menuCategory.Name,
                Description = menuCategory.Description
            };

            return View(menuCategoryToEdit);
        }

        [HttpPost]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> Edit(int id, [Bind("Id,Name,Description,Image")]
    MenuCategoryOperationViewModel menuCategory)
        {
            if (id != menuCategory.Id)
            {
                return NotFound();
            }

            if (!ModelState.IsValid) return View(menuCategory);

            var oldMenuCategory = await _context.MenuCategories.FindAsync(id);

            var menuCategoryModel = new MenuCategory
            {
                Id = menuCategory.Id,
                Name = menuCategory.Name,
                Description = menuCategory.Description
            };

            if (menuCategory.Image != null)
            {
                var pictureUrl = await _photoService.SavePhotoAsync(menuCategory.Image,
    "menuCategories", menuCategory.Name + "_image");
                menuCategoryModel.ImageUrl = pictureUrl;
            }
            else
            {
                menuCategoryModel.ImageUrl = oldMenuCategory.ImageUrl;
            }

            try
            {
```

```
                _context.Update(menuCategoryModel);
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!MenuCategoryExists(menuCategory.Id))
                {
                    return NotFound();
                }

                throw;
            }
            return RedirectToAction(nameof(Index));
        }

        public async Task<IActionResult> Delete(int? id)
        {
            if (id == null || _context.MenuCategories == null)
            {
                return NotFound();
            }

            var menuCategory = await _context.MenuCategories
                .FirstOrDefaultAsync(m => m.Id == id);
            if (menuCategory == null)
            {
                return NotFound();
            }

            return View(menuCategory);
        }

        [HttpPost, ActionName("Delete")]
        [ValidateAntiForgeryToken]
        public async Task<IActionResult> DeleteConfirmed(int id)
        {
            if (_context.MenuCategories == null)
            {
                return Problem("Entity set 'ApplicationDbContext.MenuCategories'  is
    null.");
            }
            var menuCategory = await _context.MenuCategories.FindAsync(id);
            if (menuCategory != null)
            {
                _context.MenuCategories.Remove(menuCategory);
            }

            await _context.SaveChangesAsync();
            return RedirectToAction(nameof(Index));
        }

        private bool MenuCategoryExists(int id)
        {
            return (_context.MenuCategories?.Any(e => e.Id == id)).GetValueOrDefault();
        }
    }
    using Microsoft.AspNetCore.Authorization;
    using Microsoft.AspNetCore.Mvc;
    using Microsoft.AspNetCore.Mvc.Rendering;
    using Microsoft.EntityFrameworkCore;
    using SkyService.WebApplication.Data;
    using SkyService.WebApplication.Models;
```

```
using SkyService.WebApplication.Services.PhotoService;
using SkyService.WebApplication.ViewModels;

namespace SkyService.WebApplication.Controllers;

[Authorize]
public class MenuFoodsController : Controller
{
    private readonly ApplicationDbContext _context;
    private readonly IPhotoService _photoService;

    public MenuFoodsController(ApplicationDbContext context, IPhotoService
photoService)
    {
        _context = context;
        _photoService = photoService;
    }

    public async Task<IActionResult> Index(string? searchString)
    {
        if (searchString == null)
        {
            return View(await _context.MenuFoods.ToListAsync());
        }

        var menuFoods = from m in _context.MenuFoods
                        select m;

        if (!string.IsNullOrEmpty(searchString))
        {
            menuFoods = menuFoods.Where(m => m.Name.Contains(searchString) ||
m.Description.Contains(searchString));
        }

        return View(await menuFoods.ToListAsync());
    }

    [AllowAnonymous]
    public async Task<IActionResult> Details(int? id)
    {
        if (id == null || _context.MenuFoods == null)
        {
            return NotFound();
        }

        var menuFood = await _context.MenuFoods
            .FirstOrDefaultAsync(m => m.Id == id);
        if (menuFood == null)
        {
            return NotFound();
        }

        return View(menuFood);
    }

    public IActionResult Create()
    {
        ViewData["CategoryId"] = new SelectList(_context.MenuCategories, "Id",
"Name");
        return View();
    }
```

```
    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult>
Create([Bind("Id,Name,Description,Price,Image,MenuCategoryId")]
MenuFoodOperationViewModel menuFood)
    {
        if (!ModelState.IsValid) return View(menuFood);

        ViewBag.MenuCategories = await _context.MenuCategories.ToListAsync();

        var menuFoodPictureUrl = await _photoService.SavePhotoAsync(menuFood.Image,
"menuFood", menuFood.Name + " " + menuFood.Price + "_menuFood_picture");

        var menuFoodModel = new MenuFood
        {
            Name = menuFood.Name,
            ImageUrl = menuFoodPictureUrl,
            Description = menuFood.Description,
            Price = menuFood.Price,
            MenuCategoryId = menuFood.MenuCategoryId
        };

        _context.Add(menuFoodModel);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }

    public async Task<IActionResult> Edit(int? id)
    {
        ViewData["CategoryId"] = new SelectList(_context.MenuCategories, "Id",
"Name");

        if (id == null || _context.MenuFoods == null)
        {
            return NotFound();
        }

        var menuFood = await _context.MenuFoods.FindAsync(id);
        if (menuFood == null)
        {
            return NotFound();
        }

        var menuFoodOperationViewModel = new MenuFoodOperationViewModel
        {
            Id = menuFood.Id,
            Name = menuFood.Name,
            Description = menuFood.Description,
            Price = menuFood.Price,
            MenuCategoryId = menuFood.MenuCategoryId
        };

        return View(menuFoodOperationViewModel);
    }

    [HttpPost]
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> Edit(int id,
[Bind("Id,Name,Description,Price,Image,MenuCategoryId")] MenuFoodOperationViewModel
menuFood)
    {
        if (id != menuFood.Id)
```

```
        {
            return NotFound();
        }

        var menuFoodOld = await _context.MenuFoods.FindAsync(id);

        var menuFoodModel = new MenuFood
        {
            Id = menuFood.Id,
            Name = menuFood.Name,
            Description = menuFood.Description,
            Price = menuFood.Price,
        };

        if (menuFood.Image != null)
        {
            var menuFoodPictureUrl = await
_photoService.SavePhotoAsync(menuFood.Image, "menuFood", menuFood.Name + " " +
menuFood.Price + "_menuFood_picture");
            menuFoodModel.ImageUrl = menuFoodPictureUrl;
        }
        else
        {
            menuFoodModel.ImageUrl = menuFoodOld.ImageUrl;
        }

        if (!ModelState.IsValid) return View(menuFood);
        try
        {
            _context.Update(menuFoodModel);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!MenuFoodExists(menuFood.Id))
            {
                return NotFound();
            }

            throw;
        }
        return RedirectToAction(nameof(Index));
    }

    public async Task<IActionResult> Delete(int? id)
    {
        if (id == null || _context.MenuFoods == null)
        {
            return NotFound();
        }

        var menuFood = await _context.MenuFoods
            .FirstOrDefaultAsync(m => m.Id == id);
        if (menuFood == null)
        {
            return NotFound();
        }

        return View(menuFood);
    }

    [HttpPost, ActionName("Delete")]
```

```
    [ValidateAntiForgeryToken]
    public async Task<IActionResult> DeleteConfirmed(int id)
    {
        if (_context.MenuFoods == null)
        {
            return Problem("Entity set 'ApplicationDbContext.MenuFoods'  is null.");
        }
        var menuFood = await _context.MenuFoods.FindAsync(id);
        if (menuFood != null)
        {
            _context.MenuFoods.Remove(menuFood);
        }

        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }

    private bool MenuFoodExists(int id)
    {
        return (_context.MenuFoods?.Any(e => e.Id == id)).GetValueOrDefault();
    }
}
using Microsoft.EntityFrameworkCore.Metadata;
using Microsoft.EntityFrameworkCore.Migrations;
using System;

namespace SkyService.WebApplication.Data.Migrations
{
    public partial class CreateIdentitySchema : Migration
    {
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "AspNetRoles",
                columns: table => new
                {
                    Id = table.Column<string>(nullable: false),
                    Name = table.Column<string>(maxLength: 256, nullable: true),
                    NormalizedName = table.Column<string>(maxLength: 256, nullable:
true),
                    ConcurrencyStamp = table.Column<string>(nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetRoles", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "AspNetUsers",
                columns: table => new
                {
                    Id = table.Column<string>(nullable: false),
                    UserName = table.Column<string>(maxLength: 256, nullable: true),
                    NormalizedUserName = table.Column<string>(maxLength: 256,
nullable: true),
                    Email = table.Column<string>(maxLength: 256, nullable: true),
                    NormalizedEmail = table.Column<string>(maxLength: 256, nullable:
true),
                    EmailConfirmed = table.Column<bool>(nullable: false),
                    PasswordHash = table.Column<string>(nullable: true),
                    SecurityStamp = table.Column<string>(nullable: true),
                    ConcurrencyStamp = table.Column<string>(nullable: true),
```

```
                    PhoneNumber = table.Column<string>(nullable: true),
                    PhoneNumberConfirmed = table.Column<bool>(nullable: false),
                    TwoFactorEnabled = table.Column<bool>(nullable: false),
                    LockoutEnd = table.Column<DateTimeOffset>(nullable: true),
                    LockoutEnabled = table.Column<bool>(nullable: false),
                    AccessFailedCount = table.Column<int>(nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetUsers", x => x.Id);
                });

            migrationBuilder.CreateTable(
                name: "AspNetRoleClaims",
                columns: table => new
                {
                    Id = table.Column<int>(nullable: false)
                        .Annotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn),
                    RoleId = table.Column<string>(nullable: false),
                    ClaimType = table.Column<string>(nullable: true),
                    ClaimValue = table.Column<string>(nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetRoleClaims", x => x.Id);
                    table.ForeignKey(
                        name: "FK_AspNetRoleClaims_AspNetRoles_RoleId",
                        column: x => x.RoleId,
                        principalTable: "AspNetRoles",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "AspNetUserClaims",
                columns: table => new
                {
                    Id = table.Column<int>(nullable: false)
                        .Annotation("SqlServer:ValueGenerationStrategy",
SqlServerValueGenerationStrategy.IdentityColumn),
                    UserId = table.Column<string>(nullable: false),
                    ClaimType = table.Column<string>(nullable: true),
                    ClaimValue = table.Column<string>(nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetUserClaims", x => x.Id);
                    table.ForeignKey(
                        name: "FK_AspNetUserClaims_AspNetUsers_UserId",
                        column: x => x.UserId,
                        principalTable: "AspNetUsers",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "AspNetUserLogins",
                columns: table => new
                {
                    LoginProvider = table.Column<string>(maxLength: 128, nullable:
false),
```

```
                    ProviderKey = table.Column<string>(maxLength: 128, nullable:
false),
                    ProviderDisplayName = table.Column<string>(nullable: true),
                    UserId = table.Column<string>(nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetUserLogins", x => new {
x.LoginProvider, x.ProviderKey });
                    table.ForeignKey(
                        name: "FK_AspNetUserLogins_AspNetUsers_UserId",
                        column: x => x.UserId,
                        principalTable: "AspNetUsers",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "AspNetUserRoles",
                columns: table => new
                {
                    UserId = table.Column<string>(nullable: false),
                    RoleId = table.Column<string>(nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetUserRoles", x => new { x.UserId,
x.RoleId });
                    table.ForeignKey(
                        name: "FK_AspNetUserRoles_AspNetRoles_RoleId",
                        column: x => x.RoleId,
                        principalTable: "AspNetRoles",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                    table.ForeignKey(
                        name: "FK_AspNetUserRoles_AspNetUsers_UserId",
                        column: x => x.UserId,
                        principalTable: "AspNetUsers",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
                });

            migrationBuilder.CreateTable(
                name: "AspNetUserTokens",
                columns: table => new
                {
                    UserId = table.Column<string>(nullable: false),
                    LoginProvider = table.Column<string>(maxLength: 128, nullable:
false),
                    Name = table.Column<string>(maxLength: 128, nullable: false),
                    Value = table.Column<string>(nullable: true)
                },
                constraints: table =>
                {
                    table.PrimaryKey("PK_AspNetUserTokens", x => new { x.UserId,
x.LoginProvider, x.Name });
                    table.ForeignKey(
                        name: "FK_AspNetUserTokens_AspNetUsers_UserId",
                        column: x => x.UserId,
                        principalTable: "AspNetUsers",
                        principalColumn: "Id",
                        onDelete: ReferentialAction.Cascade);
```

```
        });

    migrationBuilder.CreateIndex(
        name: "IX_AspNetRoleClaims_RoleId",
        table: "AspNetRoleClaims",
        column: "RoleId");

    migrationBuilder.CreateIndex(
        name: "RoleNameIndex",
        table: "AspNetRoles",
        column: "NormalizedName",
        unique: true,
        filter: "[NormalizedName] IS NOT NULL");

    migrationBuilder.CreateIndex(
        name: "IX_AspNetUserClaims_UserId",
        table: "AspNetUserClaims",
        column: "UserId");

    migrationBuilder.CreateIndex(
        name: "IX_AspNetUserLogins_UserId",
        table: "AspNetUserLogins",
        column: "UserId");

    migrationBuilder.CreateIndex(
        name: "IX_AspNetUserRoles_RoleId",
        table: "AspNetUserRoles",
        column: "RoleId");

    migrationBuilder.CreateIndex(
        name: "EmailIndex",
        table: "AspNetUsers",
        column: "NormalizedEmail");

    migrationBuilder.CreateIndex(
        name: "UserNameIndex",
        table: "AspNetUsers",
        column: "NormalizedUserName",
        unique: true,
        filter: "[NormalizedUserName] IS NOT NULL");
}

protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "AspNetRoleClaims");

    migrationBuilder.DropTable(
        name: "AspNetUserClaims");

    migrationBuilder.DropTable(
        name: "AspNetUserLogins");

    migrationBuilder.DropTable(
        name: "AspNetUserRoles");

    migrationBuilder.DropTable(
        name: "AspNetUserTokens");

    migrationBuilder.DropTable(
        name: "AspNetRoles");
```

```
            migrationBuilder.DropTable(
                name: "AspNetUsers");
        }
    }
}
```