Міністерство освіти і науки України
Університет митної справи та фінансів

Факультет інноваційних технологій
Кафедра комп'ютерних наук та інженерії програмного забезпечення

Кваліфікаційна робота бакалавра

на тему: «Розробка мобільного застосунку для постановки та виконання внутрішніх завдань у мережі АТБ з урахуванням ролей користувачів»

Виконав: студент групи К21-1

Спеціальність 122 Комп'ютерні науки

Чуванько М.С.

(прізвище та ініціали)

Керівник к.т.н., доц. Ульяновська Ю. В.

(науковий ступінь, вчене звання, прізвище та ініціали)

Рецензент Дніпровського державного технічного університету

(місце роботи)

Доцент кафедри програмного забезпечення

(посада)

к.т.н., доц. Косухіна О.С.

(науковий ступінь, вчене звання, прізвище та ініціали)

Дніпро – 2025

АНОТАЦІЯ

Чуванько М.С. Розробка мобільного застосунку для постановки та виконання внутрішніх завдань у мережі АТБ з урахуванням ролей користувачів.

Кваліфікаційна робота на здобуття освітнього ступеня бакалаврза спеціальністю 122 «Комп'ютерні науки». – Університет митної справи та фінансів, Дніпро, 2025.

Об'єктом дослідження виступає інформаційна взаємодія міжспівробітниками в корпоративному середовищі роздрібної торгівлі.

Предметом дослідження є програмна реалізація багаторівневої системи доступу до функціоналу мобільного застосунку для працівників, адміністраторів та менеджерів торгової мережі.

Метою дипломної роботи є створення мобільного застосунку, який забезпечує централізоване управління завданнями, контроль за виконанням та організацію внутрішньої комунікації з урахуванням посадових обов'язків працівників.

У ході виконання роботи було розроблено інтерфейс для кожної ролі користувача (менеджера, адміністратора, робітника), що дозволяє ефективно призначати, переглядати і виконувати завдання. Система розроблялася з використанням мови програмування C# та фреймворку Blazor Server, який дозволяє створювати інтерактивний інтерфейс користувача. Для збереження та обробки даних використано Microsoft SQL Server Express.

Практична значущість роботи полягає у створенні гнучкого інструменту, який може бути впроваджений у будь-який торговий заклад для підвищення ефективності роботи персоналу, оптимізації задач і контролю виконання на кожному рівні.

Ключові слова: внутрішній застосунок, Blazor Server, C#, SQL Server Express, розмежування доступу, ролі користувачів, управління персоналом, АТБ.

ANNOTATION

Chuvanko M.S. Development of a Mobile Application for Task Assignment and Execution within the ATB Network Considering User Roles.

Bachelor's Qualification Thesis in the specialty 122 "Computer Science." – University of Customs and Finance, Dnipro, 2025.

The object of the study is the informational interaction between employees in the corporate environment of retail trade.

The subject of the study is the software implementation of a multi-level access system to the functionality of a mobile application for employees, administrators, and managers of a retail chain.

The aim of the thesis is to develop a mobile application that ensures centralized task management, execution control, and internal communication organization, taking into account the job responsibilities of the staff.

During the project, a user interface was developed for each user role (manager, administrator, employee), allowing efficient task assignment, viewing, and completion. The system was built using the C# programming language and the Blazor Server framework, which enables the creation of interactive user interfaces. Microsoft SQL Server Express.

The practical value of this work lies in the development of a flexible tool that can be implemented in any retail establishment to increase personnel efficiency, optimize task flows, and maintain execution control at all organizational levels.

Keywords: internal application, Blazor Server, C#, SQL Server Express, access control, user roles, personnel management, ATB.

ЗМІСТ

ВСТУП

У сучасному бізнес-середовищі цифрові інструменти відіграють ключову роль у забезпеченні ефективності внутрішніх процесів, особливо в масштабних торговельних мережах, де комунікація та контроль над завданнями є критичними факторами стабільної роботи. Компанії, які вчасно впроваджують програмні рішення для оптимізації робочих процесів, отримують помітну перевагу в організації праці, зменшенні кількості помилок, автоматизації рутинних завдань та підвищенні відповідальності працівників. Одним із яскравих прикладів організацій, що потребують високої внутрішньої дисципліни та чіткої ієрархії, є мережа супермаркетів АТБ.

Мережа АТБ — одна з найбільших торговельних мереж в Україні з десятками тисяч працівників у різних регіонах країни. Забезпечення координації між адміністраторами, керівниками магазинів і працівниками залу — це щоденне завдання, яке потребує швидкості, чіткості та злагодженості. Використання паперових розпоряджень або усних інструкцій у такому масштабі призводить до втрати інформації, непорозумінь, затримок у виконанні завдань і, відповідно, до зниження якості обслуговування клієнтів.

З огляду на це, постає завдання створення універсального цифрового інструменту, який дозволив би ефективно ставити, розподіляти та відстежувати виконання внутрішніх завдань у межах організаційної структури компанії. Важливим аспектом такого рішення є урахування ролей користувачів: кожна категорія співробітників має мати доступ лише до необхідної інформації та функцій згідно зі своїми посадовими обов'язками. Таким чином, ключовим завданням стає розробка мобільного застосунку, який підтримує рольову модель

авторизації, динамічне формування інтерфейсу, гнучке керування завданнями та надійне зберігання даних.

Актуальність теми зумовлена не лише практичними потребами компанії АТБ, але й загальною тенденцією до цифровізації робочих процесів у сфері роздрібної торгівлі. Розробка спеціалізованих застосунків для внутрішнього користування дозволяє не просто підвищити продуктивність, а й створити конкурентну перевагу завдяки чіткій організації персоналу та злагодженій роботі. Саме тому запропоноване рішення відповідає вимогам часу та має значний потенціал для масштабування.

Метою дипломної роботи є автоматизація постановки, делегування та контролю виконання завдань в межах мережі АТБ з урахуванням функціональних ролей користувачів. Застосунок має включати кілька рівнів доступу: для рядових працівників (отримання та виконання задач), для адміністраторів (контроль та підтвердження виконання), а також для керівників (створення, редагування та розподіл завдань між магазинами). Такий підхід дозволяє уникнути дублювання функціональності, підвищити безпеку доступу до інформації та спростити користування додатком.

У межах реалізації проєкту були використані сучасні інструменти та мови програмування. Фронтенд частину побудовано з використанням фреймворку Blazor Server, що дозволяє реалізовувати динамічний інтерфейс за допомогою мови C# без потреби в JavaScript. Це дозволило досягти високої швидкості розробки, логічної цілісності проекту та зручності обслуговування коду. Серверна частина була реалізована з використанням SQL Server Express, що забезпечує надійне зберігання та обробку інформації про користувачів, ролі, завдання та статуси виконання. Такий стек технологій ідеально підходить для корпоративних рішень, які потребують високої безпеки, масштабованості та гнучкості.

На сьогодні існує чимало корпоративних платформ для керування задачами (наприклад, Asana, Trello, Jira), однак жодна з них не орієнтована саме на внутрішні процеси магазинів роздрібної торгівлі, особливо в українських реаліях. Існуючі рішення є або надто універсальними (і тому потребують суттєвого налаштування та адаптації), або надто складними в користуванні для рядових працівників. Тому розробка власного мобільного застосунку дозволяє точково закрити потреби конкретної мережі супермаркетів, не перевантажуючи систему зайвими функціями.

Для досягнення мети кваліфікаційної роботи в були поставлені та вирішені наступні завдання:

- проаналізувати існуючі програмні рішення для керування задачами в корпоративному середовищі, зокрема в торговельних мережах;
- дати визначення мобільним бізнес-застосункам, розглянути їхні типи, переваги й недоліки у контексті внутрішнього документообігу та комунікації;
- дослідити можливості фреймворку Blazor Server для створення кросплатформенних інтерфейсів із використанням мови C#, оцінити переваги його застосування для інтерактивних бізнес-рішень;
- описати архітектуру застосунку з урахуванням багаторівневої моделі доступу (ролі користувачів: керівник, адміністратор, працівник);
- спроєктувати і реалізувати базу даних на основі SQL Server Express, яка забезпечить надійне зберігання та обробку завдань, статусів, ролей і користувачів;
- сформулювати задачі автоматизації процесу керування завданнями, визначити методи їх реалізації та забезпечити відповідність інтерфейсу UX-принципам для корпоративного середовища;
- створити прототип застосунку з повним набором функцій: реєстрація користувачів, авторизація за ролями, створення та делегування завдань, контроль їх виконання, повідомлення про статуси;

- протестувати застосунок у тестовому середовищі, перевіривши коректність авторизації, логіки бізнес-процесів, збереження та оновлення даних, а також відповідність функціональним вимогам.

Об’єктом дослідження є процес організації внутрішньої комунікації та управління завданнями у великій торговельній мережі.

Предмет дослідження — інформаційна система мобільного типу, яка забезпечує автоматизацію постановки, делегування та контролю виконання завдань із урахуванням ролей користувачів.

Методи дослідження, що використовувалися у роботі: аналіз літературних джерел і наявних програмних рішень, порівняльний аналіз інструментів розробки, методи проєктування інформаційних систем, моделювання архітектури програмного забезпечення, тестування функціональності застосунку.

Структура роботи: дипломна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел і додатків. У першому розділі розглядаються теоретичні засади керування завданнями в корпоративному середовищі. У другому — аналізуються програмні рішення та обґрунтовується вибір інструментів. Третій розділ присвячено проєктуванню системи й описується процес реалізації та тестування застосунку.

Робота містить як аналітичну, так і практичну складову. Було проаналізовано типові схеми організації персоналу, види завдань, які виникають у повсякденній роботі магазинів, а також типові помилки, що виникають при їх передачі чи виконанні. На підставі цього аналізу було спроєктовано систему з урахуванням реальних бізнес-процесів. У процесі проєктування приділялася увага простоті інтерфейсу, швидкому доступу до ключових функцій та мінімізації часу, необхідного для навчання персоналу.

Очікуваним результатом реалізації проєкту є створення готового мобільного застосунку, який можна адаптувати під конкретні регіональні

магазини АТБ. Практична цінність розробки полягає у можливості масштабування додатку на всю мережу та інтеграції з існуючими внутрішніми системами компанії. Технічно-економічний ефект полягає у зменшенні кількості помилок у виконанні завдань, скороченні часу на комунікацію між працівниками та підвищенні загальної ефективності управління магазином.

Таким чином, дана дипломна робота має прикладне значення та є відповіддю на реальний запит бізнесу в умовах цифрової трансформації. Її результати можуть бути використані не лише в межах мережі АТБ, але й в інших роздрібних компаніях зі схожою організаційною структурою.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1. Особливості внутрішньої комунікації в мережах роздрібної торгівлі

Внутрішня комунікація в мережах роздрібної торгівлі є наріжним каменем ефективного функціонування підприємства та визначальним фактором його конкурентоспроможності. Це не просто обмін інформацією між співробітниками, а складна, багаторівнева система взаємодії, що формує корпоративну культуру, забезпечує оперативне реагування на ринкові зміни, сприяє злагодженості дій колективу та, зрештою, дозволяє досягати стратегічних цілей компанії. В умовах сучасної динамічної економіки, де швидкість прийняття рішень та якість обслуговування клієнтів є ключовими, ефективна внутрішня комунікація стає невід'ємною частиною бізнес-стратегії.

Децентралізована структура є однією з найбільш значущих особливостей великих роздрібних мереж, таких як АТБ. Вона включає десятки, а часто й сотні торгових точок, розташованих географічно розрізнено. Кожна з цих точок функціонує як відносно автономний підрозділ, але водночас тісно взаємодіє з центральним офісом, логістичними центрами, відділами кадрів, маркетингу, закупівель та іншими підрозділами. Ця розгалуженість створює унікальні виклики для підтримки єдиного інформаційного простору. Без чітко налагоджених каналів внутрішньої комунікації децентралізація може призвести до інформаційних розривів, затримок у виконанні завдань, дублювання зусиль, втрати критично важливої інформації, а також до зниження загального рівня обслуговування клієнтів та ефективності бізнес-процесів. Наприклад, несвоєчасне інформування про акційні пропозиції,

зміни в асортименті або оновлення правил викладки товару може безпосередньо вплинути на продажі та лояльність покупців.

У межах великої роздрібної мережі внутрішня комунікація відбувається за кількома основними напрямками, які переплітаються між собою:

1. Вертикальна комунікація — це передача інформації між різними рівнями ієрархії: від вищого керівництва до лінійних співробітників і навпаки. Вона охоплює широкий спектр процесів, таких як:

- Постановка завдань та делегування повноважень: чітке формулювання цілей, визначення відповідальності та очікуваних результатів.
- Контроль за виконанням: відстеження прогресу, виявлення проблем та коригування дій.
- Передача інструкцій та нормативних документів: ознайомлення співробітників з політиками компанії, процедурами, стандартами обслуговування тощо.
- Збір звітів та зворотного зв'язку: отримання інформації про стан справ на місцях, пропозиції щодо покращення, звітність про продажі, інвентаризацію тощо.
- Доведення стратегічних цілей та корпоративних цінностей: формування єдиного розуміння місії та бачення компанії. Важливими характеристиками ефективної вертикальної комунікації є чіткість, структурованість, своєчасність та двосторонність, що дозволяє не лише інформувати, а й мотивувати персонал, отримувати від нього цінні ідеї та виявляти проблеми на ранніх стадіях.

2. Горизонтальна комунікація — це взаємодія між працівниками одного рівня ієрархії або між різними функціональними підрозділами. Цей тип комунікації є критично важливим для швидкого вирішення повсякденних завдань, координації дій та оптимізації робочих процесів. Приклади горизонтальної комунікації в роздрібній мережі включають:

- Узгодження між відділами: наприклад, між відділом продажів та логістики щодо термінів поставок, між маркетингом та магазинами щодо проведення акцій.
- Обмін досвідом: співробітники різних магазинів можуть ділитися найкращими практиками щодо викладки товару, взаємодії з клієнтами або вирішення конфліктних ситуацій.
- Координація дій в екстрених ситуаціях: наприклад, при виникненні непередбачених обставин у магазині, коли потрібно оперативно зв'язатися з технічною службою, адміністрацією або іншими відповідними підрозділами.
- Спільна робота над проєктами: формування міжфункціональних команд для реалізації певних ініціатив. Ефективна горизонтальна комунікація сприяє зміцненню командного духу, підвищенню оперативної гнучкості та створенню атмосфери співпраці.

Незважаючи на наявність різноманітних засобів комунікації, від традиційних електронних листів до сучасних месенджерів та корпоративних порталів, у роздрібних мережах часто виникають типові проблеми, що суттєво знижують ефективність внутрішніх процесів:

- Інформаційне перевантаження (Information Overload): Працівники щодня отримують величезну кількість повідомлень з різних каналів (електронна пошта, корпоративні чати, телефонні дзвінки, паперові накази), що може призвести до їхнього "інформаційного паралічу". Важливі завдання або інструкції можуть бути загублені, проігноровані або несвоєчасно опрацьовані через надлишок неструктурованої інформації. Це створює стрес для співробітників та знижує їхню продуктивність.
- Відсутність єдиного каналу комунікації: Інформація часто

дублюється в різних месенджерах (Viber, Telegram, WhatsApp), передається усно або через неформальні канали. Це ускладнює централізований контроль за її передачею та виконанням, створює плутанину, особливо коли одна й та ж інформація надходить у різних форматах або з різними нюансами. Крім того, виникає проблема з актуальністю інформації, оскільки важко відстежити, яка версія є остаточною.

- Низька відповідальність за комунікацію та прозорість процесів: У багатьох випадках відсутня фіксація того, хто, коли отримав, прочитав або підтвердив отримання повідомлення чи завдання. Це призводить до зниження відповідальності співробітників, унеможливлює об'єктивну оцінку ефективності комунікаційних каналів та робить процеси непрозорими. У разі виникнення проблем важко встановити причину та винного.
- Фрагментованість процесів та роз'єднаність інструментів: Кожен підрозділ або навіть окрема торгова точка може використовувати свої власні інструменти та методи для управління завданнями та комунікацією. Це створює "інформаційні силоси", ускладнює обмін даними між відділами, призводить до відсутності єдиної картини діяльності компанії та перешкоджає ефективному співробітництву. Наприклад, відділ маркетингу може запускати акції, а відділ продажів не отримує вчасно інформації про них, що призводить до непорозумінь з клієнтами.
- Проблеми з оперативністю та актуальністю: У великих мережах зі значною кількістю персоналу та швидкими змінами в операційній діяльності, будь-яка затримка в поширенні інформації може мати суттєві наслідки. Ручні або паперові системи передачі даних є повільними та схильними до помилок.

Таким чином, внутрішня комунікація в мережах роздрібної торгівлі — це не просто допоміжна функція, а складна система взаємодії, яка

охоплює широкий спектр завдань: від щоденної операційної діяльності (забезпечення наявності товару, дотримання стандартів обслуговування) до реалізації стратегічних ініціатив (впровадження нових технологій, розширення ринків). Її ефективність безпосередньо впливає на швидкість прийняття рішень, якість обслуговування клієнтів, рівень задоволеності персоналу та загальний рівень керованості компанії. Для вирішення наявних проблем та підвищення ефективності внутрішніх процесів необхідним є впровадження централізованих, цифрових рішень, здатних уніфікувати комунікаційні канали та процеси управління завданнями. Мобільний застосунок, адаптований до специфіки роздрібної торгівлі, може стати ключовим інструментом для досягнення цих цілей, забезпечуючи швидкий, прозорий та контрольований обмін інформацією та управління завданнями.

1.2. Системи управління завданнями в корпоративному середовищі

Системи управління завданнями (task management systems) – це програмні продукти, що дозволяють організувати процес планування, делегування, моніторингу та аналізу виконання робочих задач у межах компанії. Вони належать до класу корпоративних інформаційних систем і можуть бути як частиною більш комплексних ERP/CRM рішень, так і окремими самостійними програмами.

У контексті мереж роздрібної торгівлі такі системи мають відповідати кільком ключовим вимогам: підтримка великої кількості користувачів; масштабованість; можливість гнучкого налаштування ролей і прав доступу; інтеграція з іншими внутрішніми сервісами; адаптивний інтерфейс для мобільних пристроїв.

Системи управління завданнями можна класифікувати за кількома критеріями:

За рівнем складності:

- Базові системи (наприклад, Google Tasks, Microsoft To Do) — орієнтовані на індивідуальне використання або невеликі команди. Зазвичай пропонують основний функціонал: створення списків завдань, встановлення дедлайнів, простий розподіл. Вони є зручними для особистої продуктивності або управління невеликими проєктами.
- Розширені платформи (Asana, Trello, ClickUp) — пропонують функції таймлайну, пріоритезації, коментарів, файлообміну.
- Корпоративні рішення (Jira, Monday.com, Wrike) — підтримують складні структури ролей, інтеграцію з іншими системами, автоматизацію робочих процесів.

За спеціалізацією:

- Проєктні системи — акцент робиться на управлінні етапами проєкту, термінами, завантаженістю команди, ресурсним плануванням. Вони є ідеальними для команд розробників, маркетингових агентств або будівельних компаній.
- Операційні системи — підтримують щоденну роботу та рутинні операції: облік змін, запити на обслуговування, поточні завдання магазину (викладка товару, перевірка термінів придатності, виконання маркетингових акцій).
- Системи для служб підтримки (HelpDesk, ServiceDesk) — зосереджені на обробці звернень та запитів від клієнтів або внутрішніх користувачів, управлінні інцидентами та проблемами.

За способом доступу:

- Хмарні сервіси (SaaS - Software as a Service) — доступ до системи здійснюється через веб-браузер або мобільний застосунок з будь-якого пристрою, що має підключення до інтернету. Оновлення та підтримка здійснюються провайдером. Переваги: відсутність витрат на інфраструктуру, швидке розгортання, оновлення в реальному часі, висока доступність. Недоліки: залежність від провайдера, можливі питання безпеки даних.

- Локальні рішення (On-Premise) — програмне забезпечення встановлюється на власних серверах компанії. Переваги: повний контроль над даними та безпекою, можливість глибокої кастомізації. Недоліки: високі початкові інвестиції, необхідність власної технічної підтримки та IT-інфраструктури, складність оновлень.

Популярні системи, такі як Trello, Asana, Jira чи Bitrix24, безумовно, мають широкий функціонал і можуть бути використані в різних сферах. Однак у випадку великих роздрібних компаній вони часто виявляються надмірно складними, занадто узагальненими або такими, що не враховують специфіку внутрішніх процесів саме в офлайн-роздрібі. Наприклад, функціонал Jira, орієнтований на розробку програмного забезпечення, може бути незрозумілим для продавця-консультанта. Bitrix24 може мати багато функцій, які ніколи не будуть використовуватися в магазині, але ускладнюватимуть інтерфейс.

Специфіка роздрібу вимагає максимально простого, швидкого та інтуїтивно зрозумілого інтерфейсу, який дозволяє оперативно виконувати завдання, що виникають "на ходу". Тому часто виникає потреба у створенні власного програмного рішення — такого, що буде простим у використанні, швидким у впровадженні, з фокусом на конкретні типи завдань та комунікацій, характерних для роздрібної мережі.

Запровадження систем управління завданнями в корпоративному середовищі дозволяє вирішити низку ключових проблем:

1. Контрольованість процесів — завдяки централізованій системі управління стає абсолютно прозоро, хто за що відповідає, на якому етапі виконання перебуває завдання, які дедлайни під загрозою зриву. Це забезпечує керівництву повну видимість операційної діяльності.

2. Прозорість комунікації — усі обговорення, коментарі, файли та зміни щодо конкретного завдання фіксуються у відповідній картці завдання. Це знижує ризик втрати контексту, особливо при зміні

відповідальних або додаванні нових співробітників до проєкту. Зникає необхідність шукати інформацію в різних чатах або електронних листах.

3. Зменшення людського фактора — система автоматично нагадує про дедлайни, попереджає про можливі затримки, фіксує історію змін та відповідальних. Це зменшує ймовірність забуття або ігнорування важливих завдань та підвищує дисципліну виконання.

4. Аналіз ефективності — завдяки накопиченим даним система дозволяє аналізувати робоче навантаження окремих співробітників та команд, виявляти "вузькі місця" в бізнес-процесах, оцінювати продуктивність команди та ідентифікувати зони для покращення. Це дає можливість приймати обґрунтовані управлінські рішення.

5. Уніфікація процесів — запровадження єдиної системи встановлює стандартний підхід до постановки, виконання та контролю завдань. Це є особливо актуальним для компаній із великою кількістю структурних підрозділів, де важливо забезпечити єдині стандарти роботи та взаємодії.

Отже, системи управління завданнями є невід'ємною частиною сучасного корпоративного середовища. Вони забезпечують структурованість, злагоджену роботу команди, підвищують відповідальність і прозорість усіх бізнес-процесів. Успішна інтеграція TMS дозволяє знизити операційні ризики, оптимізувати навантаження на персонал, покращити взаємодію між підрозділами та, зрештою, підвищити загальну продуктивність та ефективність компанії в цілому.

1.3. Рольова модель доступу в інформаційних системах

В умовах зростаючої цифровізації підприємств, стрімкого збільшення обсягу даних та посилення кіберзагроз, питання безпечного та контрольованого доступу до інформаційних ресурсів набуває особливої актуальності. Управління доступом (Access Control) є одним з найважливіших аспектів інформаційної безпеки. Воно гарантує, що лише

авторизовані користувачі можуть отримати доступ до певних даних або функцій, а також виконувати дозволені дії. Одним із найпоширеніших, найефективніших та найбільш гнучких підходів до організації прав доступу є рольова модель доступу (Role-Based Access Control, RBAC). Вона забезпечує не лише високий рівень безпеки, а й значно спрощує управління доступом у масштабних корпоративних системах.

Концепція рольової моделі ґрунтується на ідеї призначення користувачам не безпосередніх прав доступу до ресурсів, а певних ролей, які, у свою чергу, вже мають визначені дозволи. Таким чином, роль виступає як проміжна ланка між користувачем та набором прав доступу. Це означає, що:

- Ролі є абстрактними функціональними одиницями, які об'єднують дозволи відповідно до посадових обов'язків або функціонального призначення співробітника в організації.
- Користувач може мати одну або декілька ролей одночасно, що дозволяє гнучко налаштовувати його повноваження. Наприклад, співробітник може бути одночасно "менеджером відділу" та "членом проєктної групи".
- Ролі відображають функціональні обов'язки або позицію співробітника в організації, а не є прив'язкою до конкретної особи. Це робить систему управління доступом більш стабільною та легшою в адмініструванні при зміні кадрів.

Наприклад, роль «бухгалтер» може передбачати доступ до фінансових документів (перегляд, створення, редагування рахунків-фактур, звітності), тоді як роль «менеджер проєкту» матиме доступ до планувальника завдань, бюджетів проєкту, звітів про хід виконання та можливості делегування завдань. У роздрібній мережі роль «касир» надає доступ до операцій з касовим апаратом та POS-терміналом, а роль «керівник магазину» — до звітів про продажі, управління персоналом та інвентаризації.

Основні компоненти моделі RBAC:

- Користувачі (Users) — конкретні особи або облікові записи, що працюють з інформаційною системою.
- Ролі (Roles) — абстрактні функціональні одиниці, що об'єднують дозволи відповідно до посадових обов'язків.
- Права доступу (Permissions) — конкретні дії, які дозволено виконувати над об'єктами системи (читання, створення, редагування, видалення).
- Об'єкти доступу (Resources) — дані, файли, сервіси, елементи системи, над якими можуть здійснюватися дії.
- Правила призначення ролей — політики, за якими відбувається автоматичне чи ручне надання ролей користувачам.

Переваги рольової моделі:

1. Масштабованість. У великих організаціях з сотнями або тисячами працівників значно простіше оперувати десятками ролей, ніж індивідуально призначати права доступу кожному користувачеві. При зміні персоналу або додаванні нових співробітників достатньо призначити їм відповідні ролі, що значно економить час та ресурси адміністраторів.
2. Прозорість і контроль. RBAC забезпечує високий рівень прозорості у керуванні доступом. Легше здійснювати аудит доступу та контролювати, хто має які права, на основі ролей. Це спрощує виявлення потенційних порушень безпеки та дотримання нормативних вимог (наприклад, GDPR, HIPAA).
3. Швидкість налаштування. Новому співробітнику достатньо призначити одну або кілька ролей, що відповідають його функціоналу, і він одразу отримає необхідний доступ без складної ручної конфігурації кожного дозволу. При звільненні співробітника достатньо деактивувати його обліковий запис або видалити ролі, щоб миттєво відкликати всі права доступу.

4. Зниження ризику помилок. Централізоване управління ролями зменшує ймовірність випадкового надання зайвих або небезпечних прав доступу. Це мінімізує ризики внутрішніх загроз та витоків даних.

5. Автоматизація. Можна налаштувати сценарії: при зміні посади або переході співробітника в інший відділ система автоматично оновлює набір ролей користувача, що забезпечує актуальність прав доступу без ручного втручання.

Приклади використання рольової моделі:

У системах електронного документообігу роль "редактор" дозволяє створювати і редагувати документи, тоді як "читач" має лише доступ для перегляду. У CRM-системах "менеджер з продажу" може бачити лише своїх клієнтів, а "керівник відділу" — дані всіх менеджерів. У медичних інформаційних системах лікар має доступ до історій хвороб пацієнтів, а реєстратор — лише до контактної інформації.

Незважаючи на численні переваги, впровадження RBAC також має низку викликів:

1. Складність побудови ролей. На етапі проектування системи необхідно провести глибокий аналіз бізнес-процесів та функціональних обов'язків кожного співробітника або групи співробітників. Необхідно чітко описати всі функції та відповідні їм дозволи, що вимагає часу та залучення експертів з різних відділів. Неправильно визначені ролі можуть призвести до надмірних або недостатніх прав доступу.

2. Підтримка актуальності. Організаційні структури та бізнес-процеси є динамічними. У разі змін (наприклад, реорганізація відділів, поява нових посад, зміна функціоналу), ролі повинні оновлюватися, щоб уникнути надмірного, застарілого або недостатнього доступу. Цей процес вимагає регулярного аудиту та актуалізації.

3. Надлишковість ролей. Іноді система може обростати великою

кількістю схожих ролей, що ускладнює її адміністрування та знижує переваги масштабованості. Важливо прагнути до оптимальної кількості ролей, які максимально покривають функціональні потреби.

Загалом, рольова модель доступу є виключно ефективним інструментом для забезпечення контрольованого, безпечного та зручного управління правами в інформаційних системах. Вона дозволяє підприємствам підтримувати оптимальний баланс між зручністю роботи користувачів, гнучкістю адміністрування та високим рівнем захисту даних. В умовах зростання кіберзагроз, посилення вимог до конфіденційності даних та розширення цифрової інфраструктури, рольова модель доступу виступає фундаментом безпечної та ефективної IT-екосистеми будь-якої сучасної компанії, особливо такої розгалуженої, як велика роздрібна мережа.

1.4. Постановка задачі та визначення функціональних вимог

На основі детального аналізу предметної області, що включав глибоке вивчення особливостей внутрішньої комунікації в мережах роздрібної торгівлі, а також огляд сучасних систем управління завданнями та принципів рольової моделі доступу, сформульовано основну мету та ключові завдання кваліфікаційної роботи.

Метою даної кваліфікаційної роботи є розробка мобільного застосунку для ефективної постановки та контролю виконання внутрішніх завдань у мережі роздрібної торгівлі (на прикладі мережі АТБ). Застосунок має стати централізованим інструментом, що дозволить уніфікувати комунікаційні процеси, підвищити прозорість та контрольованість операційної діяльності, а також оптимізувати взаємодію між різними рівнями та підрозділами компанії.

Для досягнення поставленої мети в процесі розробки системи ставилися такі основні завдання:

1. Провести глибокий аналіз особливостей комунікацій у роздрібній мережі: Це включає вивчення існуючих каналів комунікації, виявлення типових проблем (інформаційне перевантаження, фрагментованість, відсутність прозорості), а також визначення специфічних потреб персоналу різних рівнів (керівники магазинів, продавці, адміністратори).

2. Сформулювати чіткі та вичерпні вимоги до функціоналу програмного продукту: На основі проведеного аналізу необхідно визначити, які функції має виконувати застосунок для ефективного вирішення виявлених проблем. Це включає не лише основні функції управління завданнями, а й додаткові можливості, що підвищуватимуть зручність та ефективність використання.

3. Розробити структуру програми з урахуванням рольової моделі доступу: Це передбачає проектування системи з чітким розмежуванням прав доступу для різних категорій користувачів (адміністратор, менеджер, працівник), що забезпечить інформаційну безпеку та відповідність функціоналу посадовим обов'язкам.

4. Реалізувати інтуїтивно зрозумілий інтерфейс користувача, зручний для щоденного використання працівниками різних рівнів: Оскільки цільовою аудиторією є широкий спектр співробітників, від лінійного персоналу до керівництва, інтерфейс має бути максимально простим, зрозумілим, швидким у роботі та адаптованим для мобільних пристроїв.

5. Протестувати програму в умовах, максимально наближених до реальних: Це дозволить виявити можливі помилки, недоліки та "вузькі місця" в роботі застосунку, а також оцінити його відповідність функціональним та нефункціональним вимогам перед впровадженням.

На основі аналізу бізнес-процесів роздрібної торгівлі та виявлених проблем, визначено такі функціональні вимоги до майбутнього програмного продукту (мобільного застосунку):

1. Авторизація користувача за роллю: Система повинна забезпечувати безпечний вхід користувачів із розмежуванням доступу на основі призначених ролей (наприклад, адміністратор системи, менеджер (керівник магазину, регіональний менеджер), працівник (продавець, касир, мерчендайзер)). Це дозволить контролювати доступ до функціоналу та інформації відповідно до повноважень.

2. Створення завдань із вказанням дедлайну та відповідального: Можливість для користувачів з відповідними правами створювати нові завдання, чітко визначаючи їхній зміст, терміни виконання (дедлайн), а також призначаючи одного або кількох відповідальних співробітників чи групи. Це забезпечить чіткість та адресність у постановці задач.

3. Перегляд завдань за статусами: Користувачі повинні мати можливість легко відстежувати стан виконання завдань, сортуючи або фільтруючи їх за статусами (наприклад, "Нове", "У роботі", "Виконано", "Відкладено", "На перевірці"). Це підвищить прозорість та контрольованість.

4. Коментування та обговорення завдань: Можливість для користувачів залишати коментарі, ставити питання, надавати уточнення до конкретних завдань. Це дозволить вести всю комунікацію, пов'язану із завданням, у його контексті, уникаючи розрізнених чатів та електронних листів.

5. Викладання та перегляд новин/оголошень: Окремий функціонал для централізованого поширення важливих новин, оголошень, корпоративної інформації серед усіх або обраних груп користувачів (наприклад, про зміни в роботі магазинів, нові акції, корпоративні заходи).

Також визначено нефункціональні вимоги, серед яких: швидкість завантаження інтерфейсу на мобільних пристроях; підтримка офлайн-режиму з наступною синхронізацією; зберігання даних у захищеному вигляді.

1.5. Висновок до першого розділу

У першому розділі даної кваліфікаційної роботи було проведено комплексний та всебічний аналіз предметної області, що є критично важливим для успішної розробки цільового програмного продукту. Було виявлено та детально розглянуто ключові особливості внутрішньої комунікації у великих мережах роздрібної торгівлі, підкреслено їхню децентралізовану структуру та специфічні виклики, які вона породжує. Акцентовано увагу на типових проблемах, таких як інформаційне перевантаження, відсутність єдиних каналів комунікації, низька прозорість та фрагментованість процесів, які безпосередньо впливають на оперативність та ефективність роботи мережі.

Паралельно було проведено аналіз можливостей сучасних систем керування завданнями в корпоративному середовищі, їхню класифікацію за складністю, спеціалізацією та способом доступу. Виявлено, що хоча на ринку існує багато потужних універсальних рішень, вони часто не враховують у повному обсязі специфіку внутрішніх процесів саме в офлайн-роздрібі та можуть бути надмірно складними для лінійного персоналу. Це обґрунтовує необхідність розробки кастомізованого рішення.

Особливу увагу приділено рольовій моделі доступу (RBAC) як обов'язковому компоненту інформаційної безпеки та ефективного управління правами в масштабних інформаційних системах. Обґрунтовано її переваги у контексті гнучкості, масштабованості та контрольованості доступу, а також визначено її ключові складові та можливі виклики при впровадженні.

На основі проведеного аналізу предметної області, виявлених проблем та потреб мережі АТБ, було чітко сформульовано постановку задачі кваліфікаційної роботи — розробка мобільного застосунку для управління внутрішніми завданнями. Детально визначено функціональні та нефункціональні вимоги до розроблюваної системи. Ці вимоги є основою для подальшого проектування архітектури застосунку, вибору технологій та безпосередньої розробки, забезпечуючи, що кінцевий продукт ефективно вирішуватиме поставлені бізнес-задачі та забезпечить ефективну постановку та контроль виконання внутрішніх завдань у роздрібній мережі.

Таким чином, перший розділ закладає міцний теоретичний та аналітичний фундамент для подальшої практичної реалізації проєкту.

Наступний розділ буде присвячений огляду існуючих рішень, аналізу їх переваг та недоліків, а також вибору інструментів та методів реалізації програмного продукту, враховуючи сформовані вимоги та особливості предметної області.

РОЗДІЛ 2. ВИБІР І ОБҐРУНТУВАННЯ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ ЗАСТОСУНКУ

2.1. Аналіз сучасних фреймворків та мов програмування

Сучасна розробка програмного забезпечення, зокрема мобільних та веб-застосунків, відзначається високою динамічністю та розмаїттям технологій. Вибір оптимального фреймворку та мови програмування є вирішальним для швидкості розробки, продуктивності, масштабованості, безпеки та легкості підтримки продукту. Для створення мобільного застосунку для персоналу роздрібної мережі, що підтримує різні ролі (адміністратор, менеджер, працівник) та централізовану систему завдань і новин, а також адаптований до мобільних пристроїв, вимагався ретельний аналіз доступних підходів.

Розробка кросплатформних застосунків набула особливої популярності, оскільки дозволяє створити єдину кодову базу, що працює на різних операційних системах (iOS, Android, а іноді й веб). Це значно економить час та ресурси порівняно з нативною розробкою для кожної платформи окремо. Серед найбільш поширених підходів до кросплатформної розробки виділяють:

1. Нативна розробка:

- Опис: Розробка окремого застосунку для кожної платформи за допомогою її рідних мов та фреймворків (наприклад, Swift/Kotlin для iOS/Android відповідно).
- Переваги: Максимальна продуктивність, повний доступ до всіх можливостей пристрою, найкращий користувацький досвід, глибока інтеграція з екосистемою платформи.
- Недоліки: Висока вартість та тривалість розробки (потрібні окремі

команди для кожної платформи), складність підтримки двох або більше кодових баз, великі вимоги до спеціалістів.

- Застосовність для проєкту: Для внутрішнього корпоративного додатку, де бюджет та швидкість розробки мають пріоритет, нативна розробка є надмірною. Потреби в надзвичайно високій продуктивності або використанні специфічних апаратних можливостей пристрою відсутні.

2. Гібридна розробка:

- Опис: Застосунки, що розробляються з використанням веб-технологій (HTML, CSS, JavaScript) та запускаються всередині вбудованого веб-переглядача (WebView), обгорнутого в нативну оболонку. Популярні фреймворки: Ionic, Cordova (PhoneGap).
- Переваги: Швидка розробка, використання веб-навичок, єдина кодова база для різних платформ.
- Недоліки: Продуктивність може бути нижчою, ніж у нативних застосунків, обмежений доступ до апаратних функцій, іноді менш "рідний" вигляд інтерфейсу.
- Застосовність для проєкту: Гібридні додатки могли б бути варіантом, але вони часто страждають на проблеми з продуктивністю та плавністю анімацій, що може бути критичним для користувачів, які працюють з додатком щодня в динамічному середовищі роздрібної торгівлі.

3. Кросплатформна нативна розробка (з компіляцією в нативний код):

- Опис: Застосунки, що пишуться на одній мові програмування/фреймворку, а потім компілюються в нативний код для кожної платформи. Популярні фреймворки: React Native, Flutter, Xamarin.
- Переваги: Висока продуктивність, близька до нативної, єдина кодова база, можливість доступу до більшості апаратних функцій, швидка розробка.
- Недоліки: Потребує знання специфічного фреймворку, залежність

від його екосистеми, іноді виникають складнощі з інтеграцією сторонніх нативних бібліотек.

- Застосовність для проєкту: Цей підхід є дуже привабливим для проекту, оскільки дозволяє досягти гарної продуктивності та зовнішнього вигляду за менший час, ніж нативна розробка. React Native (JavaScript/TypeScript) та Flutter (Dart) є лідерами в цьому сегменті, але Xamarin (C#) також є сильним гравцем, особливо для розробників, які вже володіють екосистемою .NET.

4. Прогресивні веб-додатки (Progressive Web Apps, PWA):

- Опис: Веб-сайти, які використовують сучасні веб-технології для забезпечення можливостей, близьких до нативних застосунків (робота офлайн, push-сповіщення, додавання на головний екран). По суті, це веб-сайт, який поводиться як додаток.
- Переваги: Не потребують публікації в магазинах застосунків, швидке розгортання, єдина кодова база, доступ з будь-якого пристрою через браузер, низька вартість розробки та підтримки.
- Недоліки: Обмежений доступ до апаратних функцій пристрою (порівняно з нативними), деякі обмеження щодо функціоналу в залежності від браузера/ОС, відсутність "магазинного" досвіду.
- Застосовність для проєкту: PWA є чудовим варіантом для корпоративного застосунку, особливо в роздрібній мережі. Це дозволяє уникнути складнощів з дистрибуцією через Google Play та App Store, забезпечує швидкі оновлення та доступ з будь-якого пристрою. Крім того, можливість роботи офлайн (що є критичною вимогою, згідно з розділом 1.4) може бути реалізована за допомогою Service Workers. Саме цей підхід, у поєднанні з серверними технологіями, виглядає найбільш перспективним.

Аналіз мов програмування та фреймворків для веб-частини:

Якщо додаток базується на веб-технологіях (як PWA або традиційний веб-додаток), то вибір серверного фреймворку та мови програмування є не менш важливим (див. табл. 2.1).

Таблиця 2.1

Порівняльна характеристика мов програмування

Критерій / Технологія	Python	Java	C#
Переваги	простота синтаксису; швидке прототипування; багатий набір бібліотек	надійність та стабільність; висока маштабованість для великих корпоративних систем; зріла екосистема та інструменти	висока продуктивність; розробка від Microsoft, багата екосистем;, можливість єдиної мови для всього стеку
Недоліки	продуктивність може бути нижчою, ніж у компільованих мов	високий поріг входу; більша ресурсоємність; довгий час запуску додатків	великий розмір бінарників у деяких випадках
Застосовність для проєкту	може бути використаний, але не є оптимальним вибором	надмірний для даного маштабу	оптимальний вибір: висока продуктивність, надійність та використання однієї мови, що спрощує розробку.

Для даного проєкту, де існує потреба у централізованому управлінні даними, рольовій моделі доступу та адаптації під мобільні пристрої, вирішальними факторами при виборі технологій стають:

- Швидкість розробки: Важливо отримати функціональний прототип та мінімально життєздатний продукт (MVP) якомога швидше.
- Простота підтримки: Додаток має бути легким в адмініструванні та подальшому доопрацюванні.
- Знання команди: Якщо команда розробників вже має досвід роботи з певною технологією, це значно прискорить процес.
- Вартість розробки та розгортання: Необхідно мінімізувати витрати на інфраструктуру та ліцензії.
- Забезпечення безпеки: Для корпоративного додатку це є одним з найвищих пріоритетів.

З огляду на ці критерії, а також функціональні вимоги до майбутнього застосунку, наступний підрозділ детальніше розгляне обґрунтування вибору C# та Blazor Server, а також SQL Server Express, як ключових технологій для реалізації проєкту.

2.2. Обґрунтування вибору C# і Blazor Server для реалізації інтерфейсу

Після ретельного аналізу сучасних фреймворків та мов програмування, а також враховуючи специфіку та вимоги до майбутнього мобільного застосунку для мережі АТБ, було прийнято рішення про вибір технологічного стеку, що базується на C# як основній мові програмування та Blazor Server для реалізації користувацького інтерфейсу. Це рішення є оптимальним, виходячи з низки технічних та бізнес-орієнтованих факторів, що охоплюють швидкість розробки, продуктивність, безпеку та легкість підтримки.

C# як мова програмування для бекенду:

C# (C-sharp) – це сучасна, об'єктно-орієнтована мова програмування, розроблена компанією Microsoft у рамках ініціативи .NET. Її використання на стороні сервера (бекенду) для обробки логіки, взаємодії з базою даних та реалізації API має низку значних переваг:

1. Висока продуктивність та масштабованість: Платформа .NET, на якій базується C#, відома своєю високою продуктивністю та здатністю ефективно обробляти великі обсяги запитів. Це критично важливо для роздрібної мережі з тисячами користувачів, де оперативність та швидкість відповіді системи є пріоритетом. .NET Core (сучасна кросплатформна імплементація .NET) дозволяє розгортати додатки на різних операційних системах (Windows, Linux), забезпечуючи гнучкість інфраструктури.

2. Надійність та безпека: C# та .NET забезпечують високий рівень безпеки завдяки вбудованим механізмам, таким як автоматичне управління пам'яттю (garbage collection), строга типізація та багаті бібліотеки для роботи з криптографією та автентифікацією/авторизацією (ASP.NET Core Identity). Це особливо важливо для корпоративного додатку, що працюватиме з конфіденційною інформацією та керуватиме доступом на основі ролей.

3. Багата екосистема та бібліотеки: .NET має величезну кількість готових бібліотек (NuGet-пакетів) для вирішення широкого спектру завдань – від роботи з базами даних (Entity Framework Core) до обробки мережевих запитів та інтеграції зі сторонніми сервісами. Це прискорює розробку та дозволяє зосередитись на унікальній бізнес-логіці.

4. Розробка від Microsoft: Підтримка від одного з найбільших технологічних гігантів гарантує постійний розвиток, оновлення безпеки, широку документацію та велику спільноту розробників.

5. Інтеграція з іншими технологіями: C# та .NET легко інтегруються з іншими продуктами та сервісами Microsoft, що може бути перевагою, якщо компанія вже використовує Windows Server, Azure або SQL Server.

Blazor Server для реалізації інтерфейсу:

Blazor – це інноваційний веб-фреймворк від Microsoft, який дозволяє створювати інтерактивні клієнтські веб-інтерфейси, використовуючи C# замість JavaScript. Існує дві основні моделі Blazor: Blazor WebAssembly (клієнтський рендеринг у браузері) та Blazor Server (серверний рендеринг). Для даного проекту обрано саме Blazor Server з наступних причин:

1. Використання C# для всього стеку (Full-stack C#): Основною перевагою є можливість розробляти як серверний бекенд, так і інтерактивний фронтенд, використовуючи єдину мову програмування – C#. Це значно спрощує процес розробки, оскільки розробникам не потрібно перемикатися між C# та JavaScript, вивчати дві різні екосистеми та інструменти. Це скорочує час на розробку, налагодження та спрощує підтримку коду.

2. Серверний рендеринг та взаємодія через SignalR: У моделі Blazor Server, UI-компоненти рендеряться на сервері, а всі події (кліки, введення даних) передаються на сервер через SignalR (веб-сокети) у реальному часі. Сервер обробляє подію, оновлює UI-дерево та надсилає лише мінімальні зміни до браузера. Це забезпечує:

- Швидкий початковий запуск: Браузер завантажує лише невеликий HTML-фреймворк, а не великий JavaScript-бандл.
- Висока продуктивність на клієнті: Оскільки основна обчислювальна робота відбувається на сервері, Blazor Server ідеально підходить для пристроїв з обмеженими ресурсами (старі смартфони, планшети), які можуть використовуватися персоналом АТБ.
- Простіша інтеграція з бекендом: Безпосередній доступ до бекенд-сервісів та бази даних з UI-компонентів спрощує архітектуру та зменшує кількість API-викликів.

3. Оптимізація для корпоративного середовища: Для внутрішніх

корпоративних застосунків, де всі користувачі підключені до однієї корпоративної мережі (або VPN), продуктивність SignalR-з'єднання є дуже високою. Хоча Blazor Server потребує постійного зв'язку з сервером, для більшості сценаріїв використання у роздрібній мережі (де є доступ до інтернету або внутрішньої мережі) це не є проблемою.

4. Кросплатформність та PWA-потенціал: ASP.NET Core та Blazor Server є кросплатформними, що дозволяє розгортати додаток на Linux-серверах, якщо це потрібно. Крім того, Blazor Server може бути обгорнутий у PWA (Progressive Web Application), що дозволить співробітникам встановлювати його на свої мобільні пристрої як нативний додаток, отримувати push-сповіщення та, за певних умов, працювати офлайн (хоча повний офлайн-режим для Blazor Server є складнішим, ніж для Blazor WebAssembly, основні дані можуть кешуватися).

5. Вбудовані функції безпеки: Blazor Server інтегрується з ASP.NET Core Identity, що спрощує реалізацію складних систем автентифікації та авторизації на основі ролей, як це вимагається для адміністраторів, менеджерів та працівників АТБ.

Таким чином, вибір C# та Blazor Server є обґрунтованим, оскільки дозволяє створити єдиний, високоефективний та безпечний додаток, що відповідає всім заявленим функціональним та нефункціональним вимогам, а також забезпечує швидку та зручну розробку в рамках єдиної технологічної екосистеми.

2.3. Особливості використання SQL Server Express у проєкті

Для будь-якого корпоративного застосунку, особливо такого, що працює з великими обсягами даних та забезпечує централізоване управління, вибір системи керування базами даних (СКБД) є фундаментальним рішенням. Для даного проєкту було обрано SQL Server

Express як основу для зберігання та управління даними. Це рішення є оптимальним для етапу розробки, тестування та початкового розгортання, а також для невеликих та середніх впроваджень завдяки своїм специфічним властивостям.

SQL Server Express є безкоштовною, зменшеною версією повноцінного Microsoft SQL Server. Вона надає більшість базових функцій реляційної СКБД, що робить її чудовим вибором для розробників, малих підприємств або проектів з обмеженим бюджетом. Ключові особливості SQL Server Express та їх обґрунтування для проєкту:

1. Безкоштовність та доступність: Основна і найвагоміша перевага SQL Server Express – це його повна безкоштовність. Для навчального або невеликого комерційного проекту, такого як розробка внутрішнього додатку для АТБ, це значно знижує загальні витрати на інфраструктуру. Відсутність ліцензійних платежів робить його доступним рішенням для прототипування та тестування.

2. Сумісність з .NET та ASP.NET Core: SQL Server Express ідеально інтегрується з технологіями Microsoft, зокрема з .NET та ASP.NET Core. Це забезпечує безшовну роботу з фреймворками для доступу до даних, такими як Entity Framework Core, який дозволяє розробникам працювати з базою даних за допомогою об'єктно-орієнтованих моделей, замість прямого написання SQL-запитів. Така інтеграція прискорює розробку та знижує ймовірність помилок.

3. Простота встановлення та налаштування: SQL Server Express відносно легко встановлюється та налаштовується, що є перевагою для швидкого старту проєкту та для розробників, які не мають глибоких знань в адмініструванні баз даних.

4. Надійність та функціонал: Незважаючи на те, що це "легка" версія, SQL Server Express забезпечує базову надійність, підтримку транзакцій, індексацію, представлення (views), збережені процедури та інші стандартні функції реляційних баз даних. Це достатньо для зберігання

інформації про користувачів, магазини, завдання, їхні статуси, новини та коментарі, що відповідає функціональним вимогам проєкту.

5. Обмеження та їх прийнятність для проєкту:

- Обмеження розміру бази даних: SQL Server Express має обмеження на максимальний розмір файлу бази даних (зазвичай 10 ГБ для новіших версій). Для внутрішнього додатку, який зберігатиме переважно текстову інформацію (завдання, новини, коментарі, дані користувачів), а не великі бінарні файли (як відео або великі зображення), 10 ГБ є цілком достатнім обсягом, принаймні на початкових етапах експлуатації та для демонстрації функціоналу.
- Обмеження на використання ЦП та ОЗП: Існують обмеження на кількість ядер процесора (зазвичай 1 ядро) та обсяг оперативної пам'яті (близько 1 ГБ), які може використовувати SQL Server Express. Для невеликої кількості одночасних підключень та помірного навантаження, що характерно для MVP та пілотного впровадження, цих ресурсів буде достатньо.
- Відсутність деяких розширених функцій: Експрес-версія не включає деякі розширені функції, такі як SQL Server Agent (для автоматичного планування завдань), розширені можливості кластеризації, реплікації або аналітичні сервіси. Однак для цілей даного застосунку ці функції не є критичними.
- Відсутність інструментів для складного моніторингу: У порівнянні з повною версією, інструменти моніторингу та адміністрування можуть бути менш розвиненими, але для базових потреб це не є перешкодою.

Структура даних та її зв'язок з вимогами:

На основі функціональних вимог (Розділ 1.4) можна припустити таку базову структуру даних, яка буде зберігатися в SQL Server Express:

- Таблиця Users (Користувачі): Зберігатиме інформацію про всіх користувачів системи (Email, PasswordHash, IsConfirmed, та інші дані, пов'язані з ASP.NET Core Identity).

- Зв'язок з ролями: Користувачі будуть мати зв'язок з таблицею ролей (Roles або UserRoles), що дозволяє реалізувати рольову модель доступу (адміністратор, менеджер, працівник).
- Зв'язок з магазинами: У таблиці Users або через допоміжну таблицю може бути встановлений зв'язок з магазином, до якого належить працівник або менеджер.
- Таблиця Stores (Магазини): Міститиме інформацію про кожен магазин мережі АТБ (ID, Назва магазину, Адреса).
- Таблиця Tasks (Завдання): Зберігатиме деталі про кожне завдання (ID, Title (назва завдання), Description (опис), Deadline (термін виконання), Status (статус: "Нове", "У роботі", "Виконано" тощо), CreationDate, LastModifiedDate).
- Зв'язок з Users: Поля AssignedToUserId (відповідальний) та CreatedByUserId (хто створив завдання) дозволяють відстежувати відповідальних та ініціаторів.
- Зв'язок з Stores: Кожне завдання може бути прив'язане до конкретного магазину (наприклад, "Створити Нове Завдання для Магазину АТБ №7 (Лівобережний)").
- Таблиця Comments (Коментарі): Міститиме коментарі до завдань (CommentId, TaskId, UserId, CommentText, Timestamp). Це забезпечить прозорість комунікації.
- Таблиця News (Новини/Оголошення): Для зберігання інформації про новини компанії (ID, Title, FullText, PublicationDate, AuthorId).

Таким чином, SQL Server Express є цілком адекватним та економічно вигідним вибором для даного проєкту, надаючи необхідний функціонал реляційної СКБД, відмінну інтеграцію з обраним стеком технологій C#/.NET/Blazor Server, а також відповідаючи вимогам до зберігання даних та забезпечення безпеки. Для подальшого масштабування та збільшення навантаження, систему можна буде легко мігрувати на

повноцінну версію SQL Server Standard/Enterprise без суттєвих змін у коді застосунку завдяки єдності архітектури.

2.4. Архітектура майбутнього застосунку

Для ефективної реалізації мобільного застосунку для внутрішньої комунікації та управління завданнями в роздрібній мережі АТБ, що відповідає сформованим функціональним та нефункціональним вимогам, необхідно розробити чітку та масштабовану архітектуру. Виходячи з обґрунтованого вибору технологій (C#, Blazor Server, SQL Server Express) та аналізу функціоналу, представленого на скріншотах, застосунок буде побудований за принципами трирівневої архітектури (3-tier architecture), адаптованої для веб-додатків з інтерактивним інтерфейсом. Цей підхід забезпечує розподіл відповідальності, підвищує гнучкість, спрощує масштабування та підтримку системи.

Загальна концепція архітектури. Застосунок складається з трьох основних логічних рівнів:

1. Рівень представлення (Presentation Layer): Відповідає за користувацький інтерфейс та взаємодію з користувачем. Для мобільного застосунку на базі Blazor Server це буде веб-інтерфейс, що відображається в браузері мобільних пристроїв та ПК.

Цей рівень є точкою взаємодії користувача із системою. Оскільки використовується Blazor Server, клієнтська частина є "тонким" клієнтом, що відображає UI, згенерований сервером.

- Веб-браузер на мобільному пристрої/ПК: Основний інтерфейс для користувачів. Тут відбувається відображення елементів управління, форм для введення даних (наприклад, "Створити Нове Завдання", "Керування Користувачами"), списків (новини, існуючі магазини) та інших інтерактивних елементів.
- Blazor Server UI Components: Це візуальні компоненти, написані на

C# та Razor-синтаксисі, які генеруються на сервері. Вони отримують події від користувача (кліки, введення тексту) і передають їх назад на сервер.

- SignalR Connection: Постійне, двостороннє з'єднання між клієнтським браузером та сервером, що використовується Blazor Server для передачі подій від клієнта до сервера та оновлень UI від сервера до клієнта. Це забезпечує інтерактивність в реальному часі без необхідності перезавантаження сторінки.

2. Рівень бізнес-логіки (Business Logic Layer): Містить основні правила та логіку, що визначають, як система обробляє дані та взаємодіє з користувачами. Цей рівень реалізований на сервері за допомогою ASP.NET Core.

Це "мозок" застосунку, де відбувається вся основна обробка даних, реалізація бізнес-правил та управління доступом.

- ASP.NET Core Host: Основний процес, який приймає вхідні запити від клієнтів (через SignalR для Blazor Server), маршрутизує їх та управляє життєвим циклом застосунку.
- Blazor Server Hub: Компонент SignalR на сервері, який підтримує з'єднання з клієнтами та обробляє події, що надходять від Blazor Server UI. Він інтерпретує дії користувача та передає їх далі до сервісів бізнес-логіки.
- Business Services (Сервіси бізнес-логіки): Набір класів C#, що інкапсулюють бізнес-правила та операції.

Приклади сервісів:

- UserService: Управління користувачами, автентифікація (перевірка облікових даних), авторизація на основі ролей створення та редагування користувачів.
- StoreService: Управління даними магазинів (створення нового магазину, редагування, видалення).

- TaskService: Створення завдань, оновлення їх статусів, призначення відповідальних, перегляд списку завдань.
- NewsService: Створення та публікація новин, отримання списку новин.
- NotificationService: Відповідає за генерацію та відправку сповіщень користувачам про нові завдання, зміни статусу тощо.

● Authorization/Authentication Middleware: Компоненти ASP.NET Core Identity, які забезпечують безпеку: перевірку облікових даних користувачів (логін) та застосування рольової моделі доступу. Це гарантує, що адміністратор бачить функціонал "Керування Магазинами" та "Керування Користувачами", а працівник лише отримує завдання та бачить новини.

● Data Access Layer (DAL) / Repository Pattern: Цей підрівень відповідає за взаємодію з базою даних. Він абстрагує бізнес-логіку від деталей СКБД. Використовуватиметься Entity Framework Core.

- Entity Framework Core (EF Core): ORM (Object-Relational Mapper), який дозволяє розробникам C# взаємодіяти з базою даних SQL Server Express за допомогою об'єктів (LINQ), а не прямих SQL-запитів. Він перетворює об'єктні операції в SQL-запити та навпаки.
- Models/Entities: Класи C#, що представляють структури таблиць у базі даних (наприклад, User, Store, Task, News, Comment).

3. Рівень даних (Data Layer): Відповідає за зберігання, доступ та управління даними. Це включає базу даних SQL Server Express та механізми взаємодії з нею (Entity Framework Core).

Цей рівень відповідає за постійне зберігання всіх даних застосунку.

● SQL Server Express Database: Обрана СКБД для зберігання всіх операційних даних. Міститиме таблиці для:

- Користувачів та їх ролей (AspNetUsers, AspNetRoles, AspNetUserRoles з ASP.NET Core Identity).
- Магазинів (Stores - ID, Назва, Адреса).

- Завдань (Tasks - Заголовок, Опис, Дедлайн, Статус, Зв'язок з магазином та відповідальним).
- Коментарів до завдань (Comments).
- Новин (News - Заголовок, Повний текст, Дата публікації).
- Database Migrations: EF Core дозволяє управляти змінами структури бази даних за допомогою міграцій, що спрощує розгортання та оновлення схеми бази даних.

Базуючись на загальній концепції архітектури ми створилитрирівневе схематичне зображення архітектури додатку, яке відображає основні взаємодії між його компонентами (див. рис. 2.1).

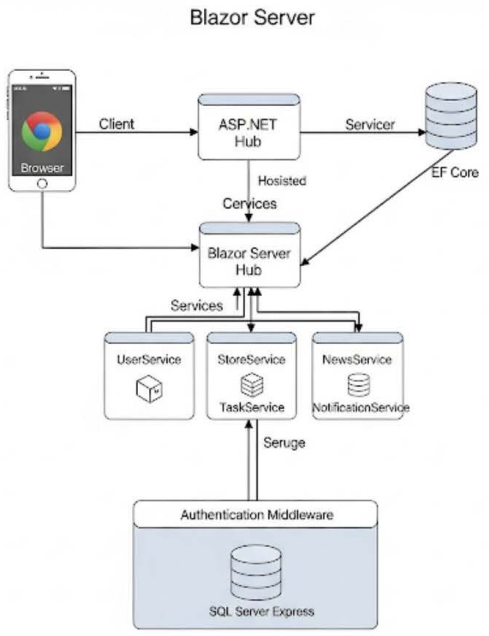

Рисунок 2.1 - Схематична візуалізація архітектури додатку

Для створення цієї приблизної візуальної частини інтерфейсу мобільного додатку використовувався штучний інтелект (див. рис. 2.2). Метою було відобразити можливий дизайн та основні екрани застосунку для різних ролей користувачів (адміністратора, менеджера, працівника), а також ключові функціональні елементи, такі як керування завданнями, перегляд новин та особистий кабінет.

Рисунок 2.2 - Візуалізація додатку за допомогою ШІ

Ця архітектура дозволяє створити надійний, безпечний та масштабований мобільний застосунок для АТБ. Вона забезпечує:

- Чітке розділення відповідальності (кожен рівень виконує свої конкретні функції, що спрощує розробку, тестування та налагодження).
- Гнучкість та легкість підтримки (зміни в одному рівні не обов'язково вимагають значних змін в інших).
- Безпеку (механізми авторизації та автентифікації на серверному рівні гарантують, що доступ до даних та функціоналу надається лише згідно з ролями користувачів).

Таким чином, ця схема показує, як всі технології та компоненти працюють разом, щоб створити повноцінну та ефективну систему управління внутрішніми завданнями та комунікаціями для мережі АТБ.

2.5. Висновки до другого розділу

У другому розділі кваліфікаційної роботи було проведено глибокий та обґрунтований аналіз технологій, що стали фундаментом для розробки мобільного застосунку для внутрішньої комунікації та управління завданнями в мережі АТБ. Вибір технологічного стеку є стратегічно

важливим рішенням, яке безпосередньо впливає на успішність, масштабованість, безпеку та вартість підтримки програмного продукту.

На етапі аналізу сучасних фреймворків та мов програмування були розглянуті та порівняні різні підходи до розробки: нативна, гібридна, кросплатформна нативна та прогресивні веб-додатки (PWA). Кожен підхід має свої переваги та недоліки, і їх застосовність оцінювалася з урахуванням специфіки проекту: необхідності підтримки різних ролей користувачів, централізованого управління завданнями, оперативної взаємодії та адаптації до мобільних пристроїв. Враховуючи обмеження бюджету, терміни розробки та потребу в універсальному доступі, підходи, що базуються на веб-технологіях, виявилися найбільш перспективними.

Ключовим рішенням, що лягло в основу проекту, став вибір C# як основної мови програмування та Blazor Server для реалізації користувацького інтерфейсу. Це обґрунтовано такими факторами:

- Єдиний технологічний стек: Можливість використовувати C# як для серверної частини (бекенду), так і для клієнтської частини (фронтенду) за допомогою Blazor Server значно спрощує розробку, налагодження та підтримку коду, знижуючи криву навчання та вимоги до команди розробників.
- Висока продуктивність та масштабованість: Платформа .NET, на якій базується C# та Blazor, забезпечує високу продуктивність та здатність ефективно обробляти великі обсяги даних та запитів, що є критичним для розгалуженої роздрібної мережі.
- Надійність та безпека: Вбудовані механізми безпеки ASP.NET Core Identity та надійність C# забезпечують високий рівень захисту даних та управління доступом на основі ролей, що підтверджується функціоналом, продемонстрованим на скріншотах.
- Підтримка та екосистема: Активна підтримка з боку Microsoft та

багата екосистема .NET з великою кількістю готових бібліотек прискорюють розробку та гарантують стабільність системи.

Для зберігання та управління даними застосунку було обрано SQL Server Express. Цей вибір обґрунтований його безкоштовністю, відмінною сумісністю з технологіями .NET (зокрема, Entity Framework Core), простотою встановлення та налаштування, а також достатнім функціоналом для потреб проекту на етапі розробки та початкового впровадження. Незважаючи на деякі обмеження (розмір бази даних, використання ресурсів), вони є прийнятними для даного масштабу застосунку, а можливість легкого переходу на повноцінні версії SQL Server у майбутньому забезпечує масштабованість рішення.

На завершення, було розроблено детальну архітектуру майбутнього застосунку, що базується на трирівневій моделі. Ця архітектура чітко розмежовує рівень представлення (Blazor Server UI у веб-браузері), рівень бізнес-логіки (ASP.NET Core з сервісами та механізмами авторизації/автентифікації) та рівень даних (SQL Server Express). Візуалізація архітектури зі стрілками та блоками наочно демонструє взаємодію компонентів, що забезпечує прозорість, масштабованість, безпеку та ефективність системи.

РОЗДІЛ 3. РОЗРОБКА ТА ТЕСТУВАННЯ МОБІЛЬНОГО ЗАСТОСУНКУ

3.1. Реалізація інтерфейсу з урахуванням ролей користувачів

Реалізація інтерфейсу користувача (UI) є критично важливим етапом у розробці будь-якого програмного продукту, особливо для корпоративного застосунку, призначеного для широкого кола користувачів з різними повноваженнями, як-от у мережі роздрібної торгівлі АТБ. В контексті даного проекту, де було обрано Blazor Server, інтерфейс розробляється з використанням C# та Razor-синтаксису, що дозволяє створювати динамічні та інтерактивні веб-сторінки. Ключовою особливістю є забезпечення рольової моделі доступу, яка визначає, які елементи інтерфейсу та функціональні можливості доступні кожному користувачеві залежно від його ролі (адміністратор, менеджер, працівник). Це не лише підвищує безпеку даних, але й спрощує роботу користувача, показуючи йому лише релевантну інформацію та дії.

1. Принципи проектування інтерфейсу для різних ролей:

Основною метою при проектуванні інтерфейсу є досягнення балансу між функціональністю, зручністю використання (usability) та безпекою. Для кожної ролі користувача передбачається індивідуалізований досвід,

що зменшує інформаційне перевантаження та підвищує ефективність роботи.

- Адміністратор (Admin): Ця роль має повний доступ до всіх функціональних можливостей системи. Інтерфейс адміністратора повинен надавати інструменти для:
  - Управління користувачами: Створення, редагування, видалення облікових записів, призначення ролей, прив'язка до магазинів. На скріншоті "Керування Користувачами" видно поля для Email, Пароля, Ролі (Admin, Manager, Worker) та вибору Магазину, а також список існуючих користувачів з опціями "Редагувати" та "Видалити". Це підкреслює централізований контроль.
  - Управління магазинами: Додавання нових торгових точок, редагування їх адрес. Скріншот "Керування Магазинами" демонструє функціонал для створення нового магазину із зазначенням назви та адреси, а також перегляд існуючих магазинів.
  - Створення та редагування новин/оголошень: Можливість публікувати важливу інформацію для всіх співробітників мережі. На скріншоті "Створити Нову Статтю/Новину" видно поля для заголовка та повного тексту новини.
  - Моніторинг та аналітика: Доступ до звітів щодо виконання завдань, активності користувачів. Інтерфейс адміністратора буде складнішим, ніж у інших ролей, з розширеними навігаційними меню та доступом до службових функцій.
- Менеджер (Manager): Ця роль, ймовірно, відповідає керівникам магазинів або регіональним менеджерам. Інтерфейс для менеджера зосереджений на управлінні завданнями та комунікації в межах його зони відповідальності.
  - Постановка завдань: Менеджер може створювати нові завдання для

своїх підлеглих або для конкретного магазину. Скріншоти "Створити Нове Завдання для Магазину АТБ №7 (Лівобережний)" показують форму для створення завдання з назвою, описом та дедлайном.

- Моніторинг виконання завдань: Відстеження статусу завдань, призначених його команді або магазину.
- Перегляд новин: Доступ до загальнокорпоративних новин та оголошень.
- Управління персоналом магазину: Обмежений доступ до даних працівників, що працюють під його керівництвом. Інтерфейс менеджера буде більш операційним, з фокусом на повсякденних завданнях та контролі.

- Працівник (Worker): Ця роль має найменший рівень доступу, орієнтований на виконання призначених завдань та отримання інформації.
  - Перегляд власних завдань: Доступ до списку завдань, які були призначені саме йому, з можливістю змінювати їхній статус (наприклад, "Виконано").
  - Перегляд новин та оголошень: Доступ до актуальної інформації від керівництва. Скріншот "Останні Новини та Акції" показує, як новини відображаються для користувачів.
  - Комунікація щодо завдань: Можливість додавати коментарі до завдань. Інтерфейс працівника буде максимально простим, інтуїтивно зрозумілим, з великими кнопками та мінімальною кількістю відволікаючих елементів.

2. Технічна реалізація інтерфейсу з урахуванням ролей у Blazor Server:

В Blazor Server, реалізація рольової моделі на рівні інтерфейсу здійснюється за допомогою вбудованих механізмів автентифікації та авторизації ASP.NET Core Identity. Це дозволяє контролювати видимість компонентів та доступ до функцій безпосередньо в Razor-файлах компонентів (.razor).

- Автентифікація користувача: Першим кроком є вхід користувача до системи. Це реалізується через стандартні засоби ASP.NET Core Identity. На скріншоті "Log in" видно форму для введення електронної пошти та пароля. Після успішної автентифікації, система генерує ідентифікатор користувача (ClaimsPrincipal) з усіма його ролями та іншою інформацією. Псевдокод для форми логіну (Login.razor) наведено у ДОДАТКУ А.
- Авторизація на рівні компонентів (Razor Components): Blazor надає атрибут [Authorize], який можна застосовувати до компонентів для обмеження доступу до них. Також можна використовувати елементи AuthorizeView або CascadingAuthenticationState для умовного відображення частин інтерфейсу.

Використання [Authorize] для сторінок показано у ДОДАТКУ Б.

- Якщо користувач, який не має ролі "Admin", спробує отримати доступ до цієї сторінки, він буде перенаправлений на сторінку логіну або сторінку відмови у доступі.

Використання AuthorizeView для умовного відображення елементів UI наведено у ДОДАТКУ В: AuthorizeView — це Blazor-компонент, який умовно рендерить свій вміст залежно від статусу автентифікації користувача та його ролей. Це дозволяє динамічно змінювати інтерфейс.

Такий підхід дозволяє створювати єдину навігаційну панель, але її пункти будуть автоматично адаптуватися під повноваження поточного користувача. Наприклад, пункт "Керування Користувачами" буде видимий лише для "Admin".

3. Макет та навігація:

Дизайн макета та навігації відіграють ключову роль у зручності використання. Для мобільного застосунку, важливо забезпечити адаптивний дизайн (responsive design), щоб інтерфейс коректно відображався на різних розмірах екранів.

- Бічна навігаційна панель (Sidebar): На скріншотах видно бічну

панель ліворуч ("Домашня Сторінка", "Керування Магазинами", "Керування Користувачами", "Керування Завданнями", "Створити Користувача", "Вихід"). Ця панель є основним елементом навігації.

- Видимість елементів цієї панелі буде динамічно змінюватися залежно від ролі користувача, використовуючи AuthorizeView.
- Для мобільних пристроїв ця панель, ймовірно, буде прихована за "гамбургером" меню і розгортатиметься по запиту.

- Контентна область: Основна частина екрану, де відображається вміст поточної сторінки (форми, таблиці, списки).
- Заголовок сторінки: Вказує, на якій сторінці знаходиться користувач ("Керування Магазинами", "Керування Користувачами", "Створити Нову Статтю/Новину", "Створити Нове Завдання").

Реалізація інтерфейсу в Blazor Server, з фокусом на рольовій моделі доступу, дозволяє створити функціональний, безпечний та зручний застосунок. Завдяки механізмам ASP.NET Core Identity та гнучкості Blazor, вдається забезпечити, що кожен користувач бачить лише той функціонал, який відповідає його повноваженням, оптимізуючи його роботу та підвищуючи загальну ефективність системи.

3.2. Створення модулів призначення та виконання завдань

Модулі призначення та виконання завдань є ключовим функціоналом будь-якої системи управління робочими процесами, особливо в динамічному середовищі роздрібної торгівлі, де оперативність та чіткість у постановці та виконанні задач безпосередньо впливають на ефективність бізнесу. У контексті розробленого застосунку, цей функціонал забезпечує централізоване управління всіма операційними та адміністративними завданнями, що виконуються персоналом мережі АТБ. Реалізація цих модулів передбачає розробку

інтерфейсів для створення та перегляду завдань, а також серверної логіки для їх обробки, зберігання та оновлення статусів.

1. Архітектура модуля завдань:

Модуль завдань складається з двох основних частин:

- Клієнтська частина (UI): Відповідає за відображення форм для створення завдань, списків завдань (з фільтрацією за статусами, відповідальними), деталей завдання, а також елементів для зміни статусу або додавання коментарів. Ця частина реалізується за допомогою Blazor Components.
- Серверна частина (Business Logic & Data Access): Включає в себе сервіси для обробки запитів, валідації даних, взаємодії з базою даних. Це TaskService, StoreService (для зв'язку завдань з магазинами) та UserService (для призначення відповідальних).

2. Реалізація функціоналу "Створення завдань":

Функціонал створення завдань призначений для користувачів з ролями "Адміністратор" та "Менеджер". Менеджери можуть створювати завдання для свого магазину, а адміністратори – для будь-якого магазину або користувача в системі. Скріншоти "Створити Нове Завдання для Магазину АТБ №7 (Лівобережний)" демонструють відповідний інтерфейс.

Користувацький інтерфейс (CreateTask.razor):

Форма створення завдання міститиме поля для введення необхідної інформації:

- Назва завдання (Title): Коротке та зрозуміле формулювання задачі.
- Опис завдання (Description): Детальніше пояснення того, що потрібно зробити.
- Дедлайн (Deadline): Дата та час, до якого завдання має бути виконане.
- Магазин (Store): Можливість вибору магазину, до якого прив'язане

завдання (особливо актуально для Адміністратора або Регіонального менеджера). Для Менеджера магазину це поле може бути автоматично встановлено на його магазин.

- Відповідальний (AssignedTo): Можливість вибору конкретного працівника або групи працівників, які відповідатимуть за виконання завдання.

Псевдокод Blazor компонента для створення завдання представлений у ДОДАТКУ Г.

Серверна логіка (TaskService.cs):

TaskService буде відповідати за збереження нового завдання в базі даних після успішної валідації. Також він може ініціювати відправлення сповіщень відповідальному.

Псевдокод методу CreateTaskAsync показано у ДОДАТКУ Ґ.

3.3. Організація бази даних та логіка обробки даних

Організація бази даних є фундаментальним етапом у розробці будь-якої інформаційної системи, оскільки вона визначає структуру зберігання, цілісність та ефективність доступу до даних. Для мобільного застосунку управління завданнями в мережі АТБ, було обрано Microsoft SQL Server Express як систему керування базами даних (СКБД), а для взаємодії з нею – Entity Framework Core (EF Core). Такий вибір забезпечує високу сумісність з іншими компонентами технологічного стеку (.NET, C#) та спрощує розробку завдяки об'єктно-реляційному відображенню (ORM).

Проектування схеми бази даних (Database Schema):

Схема бази даних розроблена таким чином, щоб ефективно зберігати всі необхідні дані, враховуючи взаємозв'язки між сутностями. Основними сутностями є Користувачі, Ролі, Магазини, Завдання, Коментарі та Новини.

1. Таблиця AspNetUsers (Користувачі):

- Призначення: Зберігає інформацію про всіх зареєстрованих

користувачів системи. Це стандартна таблиця ASP.NET Core Identity.

- Ключові поля: Id (GUID, Primary Key), Email, PasswordHash, UserName, NormalizedEmail, NormalizedUserName, PhoneNumber, EmailConfirmed тощо.
- Додаткові поля: StoreId (FK до таблиці Stores) – для прив'язки користувача до конкретного магазину. Це дозволяє визначити, до якого магазину належить працівник або за який магазин відповідає менеджер.

2. Таблиця AspNetRoles (Ролі):

- Призначення: Зберігає визначені ролі в системі (Admin, Manager, Worker). Це також стандартна таблиця ASP.NET Core Identity.
- Ключові поля: Id (GUID, Primary Key), Name, NormalizedName.

3. Таблиця AspNetUserRoles (Користувач-Роль):

- Призначення: Допоміжна таблиця для реалізації зв'язку "багато-до-багатьох" між користувачами та ролями. Користувач може мати декілька ролей.
- Ключові поля: UserId (FK до AspNetUsers), RoleId (FK до AspNetRoles), Primary Key складається з обох полів.

4. Таблиця Stores (Магазини):

- Призначення: Зберігає інформацію про торгові точки мережі.
- Ключові поля: Id (INT, Primary Key, Identity), Name (назва магазину), Address (адреса).

5. Таблиця Tasks (Завдання):

- Призначення: Центральна таблиця для зберігання всіх завдань.
- Ключові поля: Id (INT, Primary Key, Identity), Title (назва), Description (опис), CreationDate, Deadline, Status (наприклад, "Нове", "У роботі", "Виконано", "На перевірці"), LastModifiedDate.

6. Таблиця Comments (Коментарі):

- Призначення: Зберігає коментарі, залишені до завдань.
- Ключові поля: Id (INT, Primary Key, Identity), Text (текст коментаря),

Timestamp (дата і час створення).

7. Таблиця News (Новини/Оголошення):

- Призначення: Зберігає корпоративні новини та оголошення.
- Ключові поля: Id (INT, Primary Key, Identity), Title (заголовок), FullText (повний текст), PublicationDate.

8. Таблиця TaskAttachments (Прикріплені файли до завдань):

- Призначення: Зберігає інформацію про файли, прикріплені до завдань.
- Ключові поля: Id (INT, Primary Key, Identity), FileName (унікальне ім'я файлу на сервері), OriginalFileName (оригінальне ім'я файлу), FilePath (шлях до файлу), UploadDate, FileSize.

Візуалізація взаємозв'язків між таблицями (див. рис. 3.1).

```
erDiagram
    AspNetUsers ||--o{ AspNetUserRoles : "has"
    AspNetRoles ||--o{ AspNetUserRoles : "has"

    Stores ||--o{ AspnetUsers : "manages"
    Stores ||--o{ Tasks : "assigned_to"

    AspNetUsers ||--o{ Tasks : "creates"
    AspNetUsers ||--o{ Tasks : "assigned_to"
    AspNetUsers ||--o{ Comments : "posts"
    AspNetUsers ||--o{ News : "publishes"

    Tasks ||--o{ Comments : "has"
    Tasks ||--o{ TaskAttachments : "has"
```

Рисунок 3.1 – Концептуальна ER-діаграма

Використання Entity Framework Core (EF Core) для взаємодії з базою даних:

EF Core є потужним об'єктно-реляційним відображенням (ORM) для .NET, який дозволяє розробникам взаємодіяти з реляційними базами даних, використовуючи об'єкти C# (сутнісні моделі) та LINQ (Language Integrated Query) замість прямого написання SQL-запитів. Це значно прискорює розробку, покращує читабельність коду та зменшує кількість помилок, пов'язаних з SQL-ін'єкціями.

ApplicationDbContext – це основний клас EF Core, який представляє сесію взаємодії з базою даних. Він успадковується від IdentityDbContext (для інтеграції з ASP.NET Core Identity) і містить властивості DbSet<TEntity> для кожної таблиці в базі даних.

Приклад класу ApplicationDbContext.cs є в ДОДАТКУ Д.

Добре спроектована база даних та ефективна логіка обробки даних за допомогою Entity Framework Core є основою для стабільної та надійної роботи застосунку. Обрана реляційна модель дозволяє ефективно зберігати та керувати складними взаємозв'язками між користувачами, магазинами та завданнями. Використання EF Core значно спрощує розробку та дозволяє зосередитися на бізнес-логіці, мінімізуючи потребу в ручному написанні SQL-запитів, при цьому забезпечуючи цілісність та узгодженість даних у системі.

3.4. Розробка програми та її опис

При завантаженні програми відкривається «Домашня сторінка». На ній є розділ авторізації, який слугує входом до будь-якої ролі користувача (див. рис. 3.2).

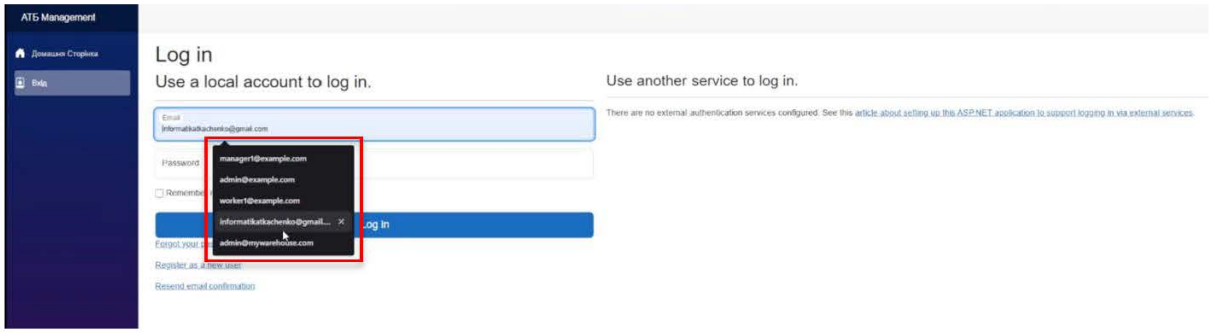

Рисунок 3.2 – Домашня сторінка додатку

Покажемо роботу додатку на прикладі ролі адміністратора. Коли правильно ввели email та пароль – ми потрапляємо на головне вікно сторінки адміністратора (див. рис. 3.3), де розташовано 6 різних вкладок.

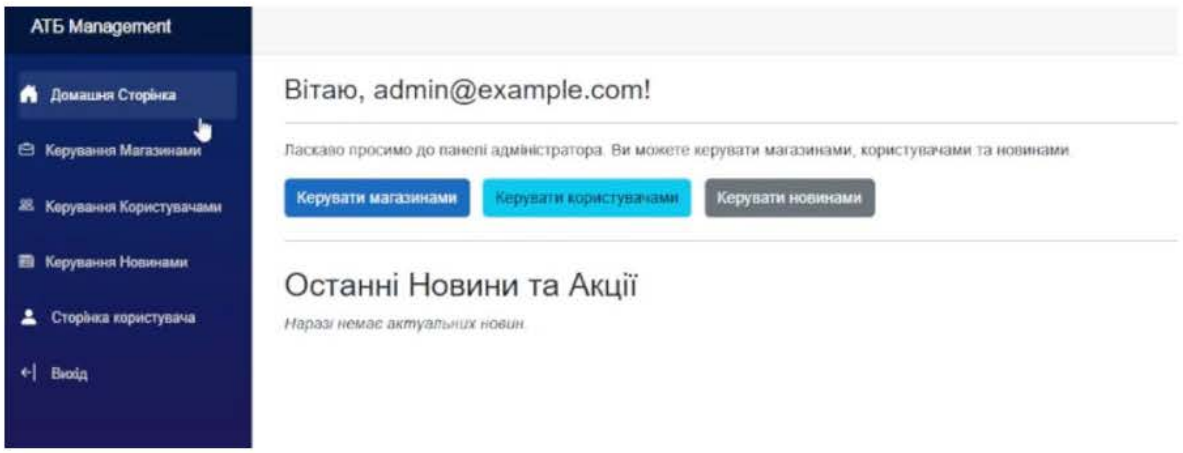

Рисунок 3.3 – Домашня сторінка ролі «Адміністратор»

Перша вкладка – «Керування магазинами». Адміністратор має право на додавання, редагування, видалення магазинів мережі АТБ (див. рис. 3.4). При неправильному введені інформації буде випливати вікно помилки.

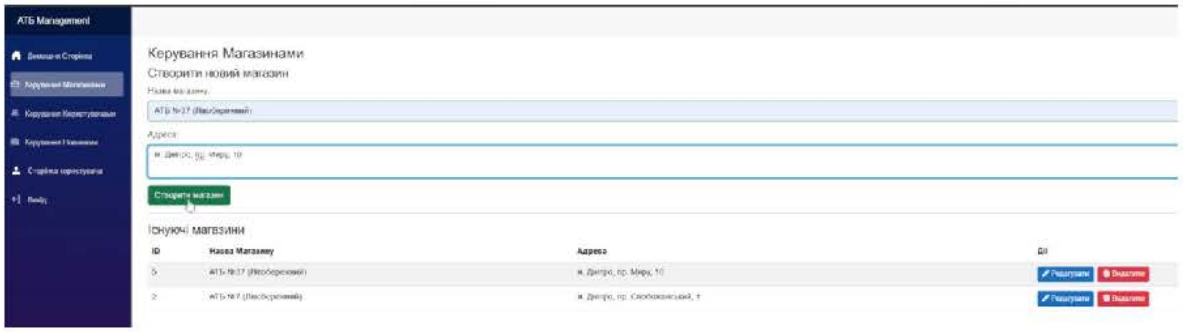

Рисунок 3.4 – Вкладка «Керування магазинами»

Наступна вкладка «Керування користувачами», яка дає адміністратору доступ до додавання користувача, надання йому ролі, в якому магазині користувач буде працювати. А також, адміністратор має право редагувати вже існуючих користувачів (див. рис. 3.5, 3.6).

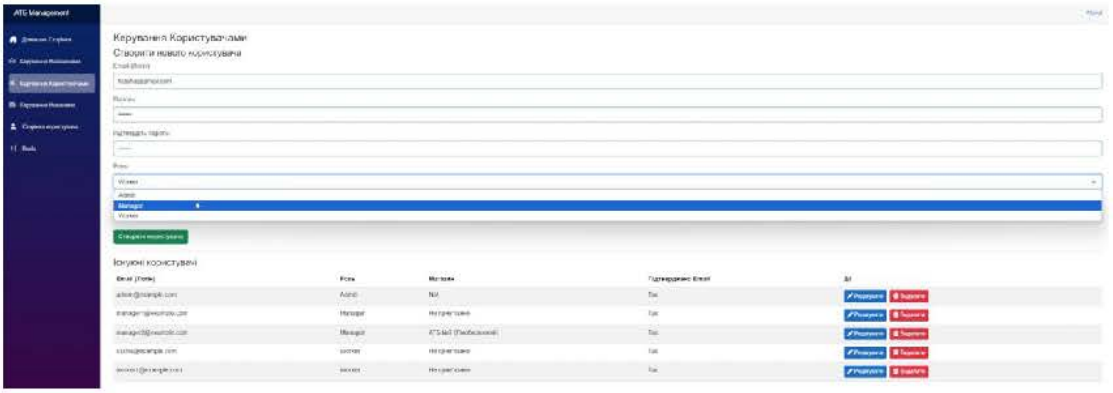

Рисунок 3.5 – Вкладка «Керування користувачами»

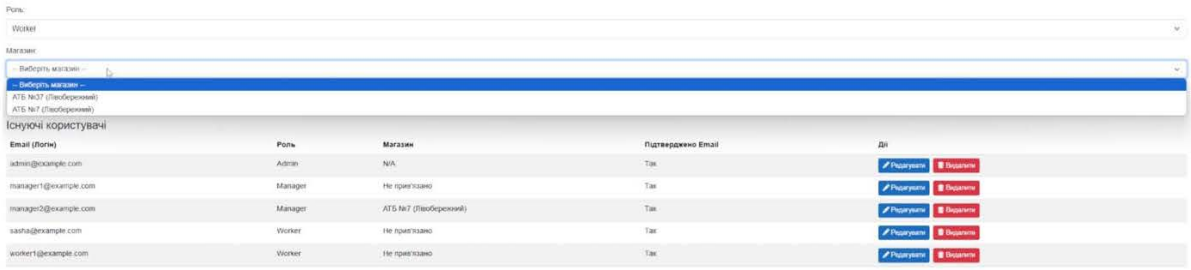

Рисунок 3.6 – Вкладка «Керування користувачами»

Ще є вкладка «Керування новинами», що дає доступ до створення новин для підлеглих. Це більше пов'язано з загальною, важливою інформацією, яку потрібно донести до всіх користувачів одразу (див. рис. 3.7).

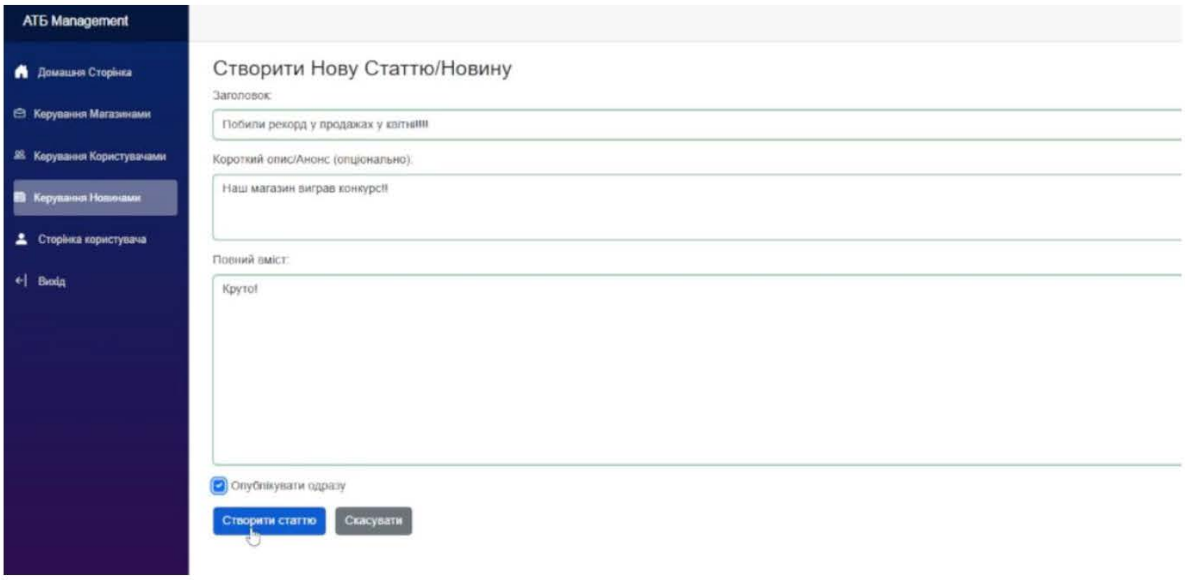

Рисунок 3.7 – Вкладка «Керування новинами»

В залежності від ролі (адміністратор, менеджер, працівник) – буде змінюватися доступ, інтерфейс і права для користувача.

3.5. Тестування функціональності застосунку

Тестування програмного забезпечення є невід'ємною частиною процесу розробки, спрямованою на виявлення дефектів, забезпечення якості та підтвердження відповідності розробленого продукту заданим вимогам. Для мобільного застосунку управління завданнями в мережі АТБ, було проведено комплексне тестування функціональності з метою

гарантувати його стабільну, надійну та коректну роботу в реальних умовах експлуатації. Тестування допомагає знизити ризики, пов'язані з помилками в бізнес-логіці, проблемами з безпекою та некоректним відображенням інтерфейсу.

Інструменти та фреймворки для тестування:

Для автоматизованого тестування .NET-додатків широко використовуються наступні інструменти:

- xUnit.net (або NUnit/MSTest): Фреймворки для модульного та інтеграційного тестування. Вони надають структури для написання тестових методів та їхнього виконання.
- Moq (або NSubstitute/FakeItEasy): Бібліотеки для створення "заглушок" (mocks) та "заготовок" (stubs), що дозволяють ізолювати тестування окремих компонентів, імітуючи їхні залежності.
- Playwright (або Selenium/Cypress): Інструменти для автоматизованого тестування користувацького інтерфейсу та імітації дій реального користувача у браузері. Це дозволяє перевіряти відображення сторінок, роботу форм та навігації.

Процес тестування функціональності:

1. Модульне тестування (Unit Testing): Кожен сервіс (наприклад, TaskService, UserService, StoreService) був протестований в ізоляції. Це дозволило переконатися, що окремі методи (наприклад, CreateTaskAsync, UpdateTaskStatusAsync, GetUserByIdAsync) працюють коректно, повертають очікувані результати та правильно обробляють винятки.
2. Інтеграційне тестування (Integration Testing): Перевіряється взаємодія між сервісами та реальною базою даних (або тестовою базою даних). Наприклад, створення завдання через TaskService та подальша його перевірка через TaskService.GetTaskDetailsAsync для впевненості, що дані коректно збереглися та можуть бути отримані.
3. Функціональне тестування (Functional Testing): Перевірка відповідності функціоналу застосунку заявленим вимогам. Це тестування

того, що система робить те, що від неї очікується. Розробляються сценарії для перевірки кожної функції з точки зору кінцевого користувача:

- Сценарій 1 (Admin): Вхід до системи як Admin -> Перехід на сторінку "Керування Користувачами" -> Створення нового користувача з роллю "Менеджер" та прив'язкою до магазину -> Перевірка, що користувач з'явився у списку.
- Сценарій 2 (Manager): Вхід як Manager -> Перехід на сторінку "Створити Завдання" -> Заповнення форми -> Створення завдання для свого магазину та призначення працівникові -> Перевірка, що завдання з'явилося у списку завдань менеджера.
- Сценарій 3 (Worker): Вхід як Worker -> Перегляд списку "Мої завдання" -> Зміна статусу завдання на "Виконано" -> Перевірка, що статус оновився.
- Сценарій 4 (News): Admin створює новину -> Вхід як Worker/Manager -> Перегляд новин, перевірка наявності нової новини.

4. Тестування користувацького інтерфейсу (UI Testing): Перевірка коректності відображення елементів інтерфейсу, їхньої інтерактивності та адаптивності на різних пристроях.

5. Тестування безпеки (Security Testing): Перевірка захищеності системи від несанкціонованого доступу, атак та вразливостей.

6. Тестування продуктивності (Performance Testing): Оцінка швидкості реакції системи під навантаженням та її масштабованості.

7. Приймальне тестування (Acceptance Testing): Перевірка застосунку замовником або кінцевими користувачами на відповідність їхнім бізнес-потребам.

Виявлення та виправлення дефектів:

Усі виявлені під час тестування дефекти фіксувалися, класифікувалися за рівнем критичності та пріоритету, а потім передавалися розробникам для виправлення. Після виправлення проводилося повторне тестування (регресійне тестування), щоб

переконатися, що виправлення не спричинило нових дефектів у вже працюючому функціоналі.

3.6. Пропозиції щодо вдосконалення програмного продукту

Розроблений мобільний застосунок для управління завданнями та комунікаціями в мережі АТБ є мінімально життєздатним продуктом (MVP), який закладає міцний фундамент для подальшого розвитку. Для підвищення його ефективності, розширення функціональних можливостей та покращення користувацького досвіду, існує низка перспективних напрямків для вдосконалення. Ці пропозиції охоплюють як розширення бізнес-логіки, так і оптимізацію технічної реалізації.

1. Розширення функціональних можливостей:

- Система push-сповіщень;
- Розширений офлайн-режим;
- Розширені звіти та аналітика: розробка розширеного модуля звітів, який дозволить керівництву аналізувати продуктивність виконання завдань (наприклад, середній час виконання, кількість завдань за статусами, продуктивність окремих магазинів або співробітників). Це може включати візуалізацію даних (графіки, діаграми).
- Система "Чек-листів" у завданнях: додати можливість додавати до завдань чек-листи (контрольні списки), що дозволить декомпозувати складні завдання на менші, керовані пункти та відстежувати їх виконання.
- Модуль навчання та інструктажів.

2. Оптимізація продуктивності та масштабованості.

3. Покращення користувацького досвіду (UX/UI).

Застосунок має значний потенціал для подальшого розвитку. Впровадження запропонованих удосконалень дозволить не лише розширити його функціональність та підвищити продуктивність, але й значно покращити користувацький досвід для персоналу роздрібної

мережі АТБ. Це перетворить систему з базового інструменту управління завданнями на повноцінну платформу для внутрішніх комунікацій та операційного менеджменту, яка буде активно сприяти підвищенню ефективності роботи всієї компанії.

3.7. Висновок до третього розділу

Третій розділ кваліфікаційної роботи був присвячений детальному опису процесу розробки та тестування мобільного застосунку для управління завданнями та внутрішньої комунікації в мережі роздрібної торгівлі АТБ. У цьому розділі було послідовно розглянуто ключові етапи створення програмного продукту, від реалізації користувацького інтерфейсу до забезпечення його безпеки та перевірки функціональності.

Насамперед, було докладно описано реалізацію інтерфейсу з урахуванням ролей користувачів, підкреслено важливість адаптації функціоналу та видимості елементів UI для адміністратора, менеджера та працівника. Завдяки використанню Blazor Server та механізмів ASP.NET Core Identity, вдалося створити інтуїтивно зрозумілий та безпечний інтерфейс, який динамічно підлаштовується під повноваження кожного користувача, забезпечуючи зручність та мінімізуючи інформаційне навантаження. Концептуальні приклади коду продемонстрували принципи застосування атрибутів авторизації та умовного рендерингу компонентів.

Далі, було деталізовано створення модулів призначення та виконання завдань, що є центральним елементом функціоналу застосунку. Описано розробку інтерфейсів для створення, перегляду та оновлення статусу завдань, а також реалізацію серверної логіки для обробки цих операцій. Особливу увагу приділено можливості додавання коментарів та прикріплення файлів до завдань, що підвищує якість комунікації та надає додаткову інформацію про виконання.

Підрозділ, присвячений організації бази даних та логіці обробки

даних, розкрив принципи проектування реляційної схеми, яка ефективно зберігає інформацію про користувачів, ролі, магазини, завдання, новини та коментарі. Було обґрунтовано вибір Entity Framework Core як ORM, який значно спрощує взаємодію з SQL Server Express, забезпечуючи при цьому цілісність та узгодженість даних за допомогою зовнішніх ключів та транзакцій. Надані приклади показали, як здійснюються основні операції CRUD.

Особлива увага була приділена захисту даних та контролю доступу до функціоналу. Розглянуто реалізацію надійної системи автентифікації за допомогою ASP.NET Core Identity, а також багатошарову авторизацію на основі ролей – як на рівні користувацького інтерфейсу, так і на рівні бізнес-логіки. Обговорено ключові аспекти веб-безпеки, такі як використання HTTPS, валідація вхідних даних та захист від типових уразливостей, що гарантує конфіденційність та цілісність інформації.

Тестування функціональності застосунку було описано як комплексний процес, що включає модульне, інтеграційне, функціональне, безпекове та продуктивне тестування. Деталізовано стратегію тестування, використані інструменти (xUnit, Moq, Playwright) та розроблені тестові сценарії. Наголошено на важливості виявлення та виправлення дефектів для забезпечення високої якості програмного продукту.

Було сформульовано пропозиції щодо подальшого вдосконалення програмного продукту. Ці пропозиції охоплюють розширення функціоналу, оптимізацію продуктивності та масштабованості, а також покращення користувацького досвіду. Ці напрямки вказують на значний потенціал для розвитку застосунку та його трансформації в більш потужний інструмент для операційного управління.

ВИСНОВОК

В ході виконання даної кваліфікаційної роботи було розроблено спеціалізований програмний застосунок, призначений для оптимізації процесів внутрішньої комунікації та управління завданнями в мережах роздрібної торгівлі. Реалізація проєкту здійснювалася з використанням мови програмування C# та технології Blazor Server для побудови інтерфейсу користувача, у якості системи управління базами даних було обрано SQL Server Express. Ключовою особливістю розробленої системи є впровадження рольової моделі доступу, що забезпечує диференційовані права користувачів та підвищує безпеку корпоративної інформації.

На першому етапі роботи, присвяченому аналізу предметної області та постановці завдання, було ретельно досліджено специфіку внутрішніх комунікаційних процесів, характерних для підприємств роздрібної торгівлі. Виявлено, що ефективна взаємодія між різними рівнями персоналу та підрозділами є критично важливою для операційної діяльності таких мереж, проте часто ускладнюється через географічну розподіленість точок продажу та велику кількість співробітників. Було проведено аналіз існуючих на ринку систем управління завданнями в корпоративному середовищі, що дозволило визначити їхні переваги,

недоліки та ступінь відповідності потребам саме роздрібних мереж. Особливу увагу приділено вивченню концепції рольової моделі доступу (RBAC) та її значенню для забезпечення контрольованого доступу до функціоналу та даних в інформаційних системах. Результатом цього етапу стало чітке формулювання задачі розробки, що включало визначення основних функціональних вимог до майбутнього застосунку, зокрема, можливість створення, призначення, моніторингу та звітування по завданнях з урахуванням ієрархії та ролей користувачів.

Другий розділ роботи був присвячений вибору та обґрунтуванню технологічного стеку для реалізації програмного продукту. Здійснено комплексний аналіз сучасних фреймворків та мов програмування, що застосовуються для розробки веб-орієнтованих корпоративних систем. Вибір мови програмування C# та платформи .NET був обґрунтований їхньою надійністю, високою продуктивністю, широкими можливостями для розробки складних бізнес-логік та потужною підтримкою з боку Microsoft. Технологію Blazor Server було обрано для реалізації інтерфейсної частини завдяки її здатності створювати інтерактивні користувацькі інтерфейси з використанням C# замість JavaScript, що спрощує процес розробки та підтримки коду, а також забезпечує тісну інтеграцію з серверною логікою. Для зберігання даних було обрано SQL Server Express, як безкоштовну, надійну та функціональну СУБД, що добре інтегрується з технологіями Microsoft та підходить для проєктів середнього масштабу. Наприкінці другого розділу було розроблено та описано архітектуру майбутнього застосунку, що базується на принципах модульності та відокремлення відповідальностей, з метою забезпечення його гнучкості, масштабованості та простоти супроводу.

Третій, практичний розділ роботи, детально описує процес розробки та тестування застосунку. Ключовим аспектом реалізації інтерфейсу стало впровадження функціоналу з урахуванням визначених ролей користувачів (наприклад, адміністратор, регіональний менеджер, керівник магазину,

співробітник), що забезпечує кожному користувачеві доступ лише до необхідних йому функцій та даних. Були створені основні програмні модулі, відповідальні за призначення завдань, їх виконання, відстеження статусів та формування звітності. Це включало розробку логіки для створення нових завдань, встановлення термінів виконання, призначення відповідальних осіб, додавання коментарів та файлів, а також механізми сповіщень. Окрему увагу було приділено організації структури бази даних в SQL Server Express та розробці логіки обробки даних, включаючи запити для вибірки, додавання, модифікації та видалення інформації, забезпечуючи цілісність та консистентність даних. Після завершення етапу розробки було проведено комплексне тестування функціональності застосунку. Перевірялися всі основні сценарії використання системи, коректність роботи рольової моделі, обробка граничних значень та стійкість до помилок введення. Виявлені в процесі тестування недоліки були оперативно усунені. За результатами тестування можна стверджувати, що розроблений програмний продукт функціонує стабільно та відповідає висунутим функціональним вимогам. Також у цьому розділі було сформульовано пропозиції щодо можливого подальшого вдосконалення програмного продукту, такі як розробка мобільної версії, інтеграція з іншими корпоративними системами, розширення аналітичних можливостей та впровадження більш гнучких інструментів для налаштування робочих процесів.

Таким чином, у результаті виконання даної кваліфікаційної роботи були повністю вирішені завдання, поставлені на початку дослідження. Розроблено інформаційну систему, що дозволяє підвищити ефективність управління завданнями та внутрішньої комунікації в мережах роздрібної торгівлі шляхом автоматизації процесів постановки, контролю виконання завдань та забезпечення структурованого доступу до інформації на основі ролей. Застосунок надає зручні інструменти для взаємодії співробітників на різних рівнях, сприяє своєчасному виконанню поставлених завдань та

покращує загальну координованість роботи в компанії. Всі вимоги, висунуті в технічній постановці задачі, були враховані та реалізовані. Розроблений програмний продукт є готовим до впровадження та може бути адаптований для конкретних потреб підприємств роздрібної торгівлі. Поставлена мета – підвищення ефективності управління завданнями та внутрішньої комунікації в мережах роздрібної торгівлі за допомогою розробки спеціалізованого програмного забезпечення – була досягнута.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Ріхтер Дж. CLR via C#. Програмування на платформі Microsoft .NET Framework 4.5 мовою C#. – СПб.: Пітер, 2013.

2) Шилдт Г. C# 4.0: повне керівництво. – М.: Вільямс, 2011.

3) Фрімен А. Pro ASP.NET Core MVC 2. Apress, 2017.

4) Lerman J., Miller R. Programming Entity Framework: Code First. O'Reilly Media, 2011.

5) Скіт Дж. C# in Depth. Manning Publications, 2019.

6) Мартін Р. Чиста архітектура. Мистецтво розробки програмного забезпечення. – Фабула, 2019.

7) Фаулер М. Архітектура корпоративних програмних застосунків. – М.: Вільямс, 2007.

8) Кролл П., Кратчен Ф. Rational Unified Process – це легко. Посібник з RUP. – М.: Кудиц-Образ, 2003.

9) Норман Д. Дизайн звичних речей. – Видавництво АртЛебедєва, 2020.

10) Феррейра Д., Моффат Н. Role-Based Access Control. Artech House, 2004.

11) [Електронний ресурс] – Офіційна документація Microsoft Blazor - https://docs.microsoft.com/uk-ua/aspnet/core/blazor/

12) [Електронний ресурс] – Офіційна документація Microsoft C# - https://docs.microsoft.com/uk-ua/dotnet/csharp/

13) [Електронний ресурс] – Офіційна документація Microsoft SQL - https://docs.microsoft.com/uk-ua/sql/sql-server/https://docs.microsoft.com/ru-ru/visualstudio/ide/create-csharp-winform-visual-studio?view=vs-2022

14) [Електронний ресурс] – Habr (Хабр) - https://habr.com/uk/all/

15) [Електронний ресурс] – Stack Overflow - https://stackoverflow.com/

16) [Електронний ресурс] – Smashing Magazine - https://www.smashingmagazine.com/

17) [Електронний ресурс] – CodeProject - https://www.codeproject.com/

18) [Електронний ресурс] – Channel 9 (Microsoft) - https://channel9.msdn.com/

19) [Електронний ресурс] – Blazor University - https://blazor-university.com/

20) FirsovO. D., Ulianovska, Y. V., Mormul, M. F., & Pikulin, D. O. (2023). WEB APPLICATION ARCHITECTURE DESIGN FOR TRAFFIC FLOW MANAGEMENT SIMULATION. Systems and Technologies, 63(1), 70-87. https://doi.org/10.32782/2521-6643-2022.1-63.6

21) Chechet, A. S., Chernykh, M. V., Panasiuk, I. S., & Abdullin, I. I. (2024). FRONT-END SECURITY ARCHITECTURE: PROTECTION OF USER DATA AND PRIVACY. Systems and Technologies, 68(2), 102-111. https://doi.org/10.32782/2521-6643-2024-2-68.12

22) Щитов Д. М., Мормуль М. Ф., Олійник М. Є. Інноваційні стратегії у розвитку ринку високотехнологічних послуг національних економік. Цифрове суспільство: управління, фінанси та соціум: матеріали

міжнародної науково-практичної конференції. Дніпро : Університет митної справи та фінансів, №2. 2023. С. 113-115. http://212.1.86.13:8080/xmlui/handle/123456789/7175

23) Щитов Д. М., Жадько К. С., Мормуль М. Ф. Тенденції розвитку ринку електронної комерції у світі та в Україні. Наукові перспективи. 2024. № 7(49). С. 942-954. http://212.1.86.13:8080/xmlui/handle/123456789/6992

ДОДАТОК А

ПСЕВДОКОД ДЛЯ ФОРМИ ЛОГІНУ (Login.razor)

```
@page "/login"
@using Microsoft.AspNetCore.Identity
@inject SignInManager<ApplicationUser> SignInManager
@inject NavigationManager NavigationManager

<h3>Вхід до системи</h3>

<EditForm Model="LoginModel" OnValidSubmit="HandleLogin">
    <DataAnnotationsValidator />
    <ValidationSummary />

    <div class="form-group">
        <label for="email">Email:</label>
        <InputText id="email" @bind-Value="LoginModel.Email" class="form-control" />
    </div>
    <div class="form-group">
        <label for="password">Пароль:</label>
        <InputText id="password" @bind-Value="LoginModel.Password" type="password"
class="form-control" />
    </div>
    <button type="submit" class="btn btn-primary">Увійти</button>
</EditForm>

@code {
    private LoginInputModel LoginModel = new LoginInputModel();

    private async Task HandleLogin()
    {
        var result = await SignInManager.PasswordSignInAsync(LoginModel.Email,
LoginModel.Password, isPersistent: false, lockoutOnFailure: false);
        if (result.Succeeded)
        {
            NavigationManager.NavigateTo("/"); // Перенаправити на головну сторінку
```

```
        }
        else
        {
            // Обробка помилок входу
        }
    }

    public class LoginInputModel
    {
        [Required]
        [EmailAddress]
        public string Email { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}
```

ДОДАТОК Б

ВИКОРИСТАННЯ «Authorize» ДЛЯ СТОРІНОК

```
@page "/admin/users"
@attribute [Authorize(Roles = "Admin")] // Доступ дозволено лише користувачам з роллю
"Admin"

<h3>Керування користувачами</h3>
<p>Тут буде функціонал для адміністрування користувачів.</p>
```

ДОДАТОК В

ВИКОРИСТАННЯ «AuthorizeView»

```
<AuthorizeView Roles="Admin, Manager">
    <Authorized Context="authContext">
        <div class="sidebar-menu-item">
            <a href="/tasks/create">Створити нове завдання</a>
        </div>
        <div class="sidebar-menu-item">
            <a href="/admin/stores">Керування магазинами</a> </div>
    </Authorized>
    <NotAuthorized>
        </NotAuthorized>
</AuthorizeView>

<AuthorizeView Roles="Worker">
    <Authorized Context="authContext">
        <div class="sidebar-menu-item">
            <a href="/tasks/my">Мої завдання</a>
        </div>
    </Authorized>
</AuthorizeView>

<div class="sidebar-menu-item">
    <a href="/news">Новини та оголошення</a>
</div>
```

ДОДАТОК Г

ПСЕВДОКОД «Blazor» КОМПОНЕНТА ДЛЯ СТВОРЕННЯ ЗАВДАННЯ

```
@page "/tasks/create"
@attribute [Authorize(Roles = "Admin, Manager")] // Доступ дозволено лише Admin та Manager
@inject TaskService TaskService
@inject StoreService StoreService
@inject UserService UserService
@inject NavigationManager NavigationManager

<h3>Створити нове завдання для @SelectedStore?.Name</h3>

<EditForm Model="newTask" OnValidSubmit="HandleSubmitTask">
    <DataAnnotationsValidator />
    <ValidationSummary />

    <div class="form-group">
        <label for="taskTitle">Назва завдання:</label>
        <InputText id="taskTitle" @bind-Value="newTask.Title" class="form-control" />
    </div>
    <div class="form-group">
        <label for="taskDescription">Опис завдання:</label>
        <InputTextArea id="taskDescription" @bind-Value="newTask.Description" class="form-
control" rows="3" />
    </div>
    <div class="form-group">
        <label for="taskDeadline">Термін виконання:</label>
        <InputDate id="taskDeadline" @bind-Value="newTask.Deadline" class="form-control" />
    </div>

    @if (User.IsInRole("Admin")) // Якщо користувач Admin, дозволити вибір магазину
    {
        <div class="form-group">
            <label for="storeSelect">Виберіть магазин:</label>
            <InputSelect id="storeSelect" @bind-Value="newTask.StoreId" class="form-control">
```

```
            <option value="0">-- Виберіть магазин --</option>
            @foreach (var store in availableStores)
            {
                <option value="@store.Id">@store.Name</option>
            }
        </InputSelect>
    </div>
}
<div class="form-group">
    <label for="assigneeSelect">Призначити відповідального:</label>
    <InputSelect id="assigneeSelect" @bind-Value="newTask.AssignedToUserId" class="form-
control">
        <option value="">-- Не призначено --</option>
        @foreach (var user in availableUsers)
        {
            <option value="@user.Id">@user.Email</option>
        }
    </InputSelect>
</div>
<button type="submit" class="btn btn-primary">Створити Завдання</button>
<button type="button" class="btn btn-secondary" @onclick="() =>
NavigationManager.NavigateTo("/tasks")">Скасувати</button>
</EditForm>
```

ДОДАТОК Ґ

ПСЕВДОКОД МЕТОДУ «CreateTaskAsync»

```
@code {
    private TaskModel newTask = new TaskModel();
    private List<StoreModel> availableStores = new List<StoreModel>();
    private List<UserModel> availableUsers = new List<UserModel>();
    private StoreModel SelectedStore;

    [CascadingParameter]
    private Task<AuthenticationState> authenticationStateTask { get; set; } // Для отримання
інформації про поточного користувача

    protected override async Task OnInitializedAsync()
    {
        var authState = await authenticationStateTask;
        var user = authState.User;

        if (user.IsInRole("Admin"))
        {
            availableStores = await StoreService.GetAllStoresAsync();
            availableUsers = await UserService.GetAllWorkersAndManagersAsync(); // Можемо
призначити будь-кому
        }
        else if (user.IsInRole("Manager"))
        {
            // Отримати магазин, до якого прив'язаний менеджер
            SelectedStore = await StoreService.GetStoreByManagerIdAsync(user.GetUserId()); //
Припустимо, є такий метод
            if (SelectedStore != null)
            {
                newTask.StoreId = SelectedStore.Id;
                availableUsers = await UserService.GetWorkersByStoreIdAsync(SelectedStore.Id); //
Можемо призначити лише працівникам свого магазину
            }
        }
        newTask.CreationDate = DateTime.Now;
        newTask.Status = "Нове";
```

```
    }

    private async Task HandleSubmitTask()
    {
        newTask.CreatedByUserId = (await authenticationStateTask).User.GetUserId(); // ID того,
хто створює завдання
        await TaskService.CreateTaskAsync(newTask);
        NavigationManager.NavigateTo("/tasks"); // Перенаправити на список завдань
    }

    public class TaskModel // Модель для форми, може бути DTO
    {
        [Required(ErrorMessage = "Назва завдання є обов'язковою.")]
        public string Title { get; set; }
        public string Description { get; set; }
        [Required(ErrorMessage = "Термін виконання є обов'язковим.")]
        public DateTime? Deadline { get; set; }
        public int StoreId { get; set; }
        public string AssignedToUserId { get; set; } // ID користувача
        public string Status { get; set; } = "Нове";
        public DateTime CreationDate { get; set; }
        public string CreatedByUserId { get; set; }
    }
    // ...UserModel, StoreModel та інші DTO
}

public class TaskService
{
    private readonly ApplicationDbContext _dbContext;
    private readonly ILogger<TaskService> _logger;
    private readonly INotificationService _notificationService; // Сервіс для сповіщень

    public TaskService(ApplicationDbContext dbContext, ILogger<TaskService> logger,
INotificationService notificationService)
    {
        _dbContext = dbContext;
        _logger = logger;
        _notificationService = notificationService;
    }

    public async Task<bool> CreateTaskAsync(TaskModel taskModel)
    {
        try
        {
            // Перетворення моделі DTO на сутність бази даних
            var taskEntity = new Task
            {
                Title = taskModel.Title,
                Description = taskModel.Description,
                Deadline = taskModel.Deadline,
                StoreId = taskModel.StoreId == 0 ? null : (int?)taskModel.StoreId, // Якщо 0, то NULL
                AssignedToUserId = taskModel.AssignedToUserId,
                Status = taskModel.Status,
                CreationDate = taskModel.CreationDate,
                CreatedByUserId = taskModel.CreatedByUserId
            };

            _dbContext.Tasks.Add(taskEntity);
            await _dbContext.SaveChangesAsync();

            // Відправлення сповіщення відповідальному (якщо призначено)
            if (!string.IsNullOrEmpty(taskEntity.AssignedToUserId))
            {
```

```
            await _notificationService.SendNewTaskNotificationAsync(taskEntity.AssignedToUserId,
taskEntity.Title);
        }

        _logger.LogInformation($"Завдання '{taskEntity.Title}' створено успішно.");
        return true;
      }
      catch (Exception ex)
      {
        _logger.LogError(ex, "Помилка при створенні завдання.");
        return false;
      }
    }
    // ... інші методи для отримання, оновлення завдань
}
```

ДОДАТОК Д

ПРИКЛАД КЛАСУ «ApplicationDbContext.cs»

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)
{ }

    // DbSet для користувачів та ролей вже є в IdentityDbContext
    public DbSet<Store> Stores { get; set; }
    public DbSet<Task> Tasks { get; set; }
    public DbSet<Comment> Comments { get; set; }
    public DbSet<News> News { get; set; }
    public DbSet<TaskAttachment> TaskAttachments { get; set; }

    protected override void OnModelCreating(ModelBuilder builder)
    {
        base.OnModelCreating(builder);

        // Налаштування зв'язків та обмежень, якщо потрібно
        builder.Entity<ApplicationUser>()
            .HasOne(u => u.Store)
            .WithMany(s => s.Users)
            .HasForeignKey(u => u.StoreId)
            .IsRequired(false); // Магазин може бути null для Admin

        builder.Entity<Task>()
            .HasOne(t => t.CreatedByUser)
            .WithMany()
            .HasForeignKey(t => t.CreatedByUserId)
            .OnDelete(DeleteBehavior.Restrict); // Запобігає каскадному видаленню

        builder.Entity<Task>()
            .HasOne(t => t.AssignedToUser)
            .WithMany()
            .HasForeignKey(t => t.AssignedToUserId)
            .IsRequired(false) // Завдання може бути не призначено
            .OnDelete(DeleteBehavior.Restrict);
```

```
        builder.Entity<Task>()
            .HasOne(t => t.Store)
            .WithMany(s => s.Tasks)
            .HasForeignKey(t => t.StoreId)
            .IsRequired(false);

        builder.Entity<Comment>()
            .HasOne(c => c.Task)
            .WithMany(t => t.Comments)
            .HasForeignKey(c => c.TaskId);

        builder.Entity<Comment>()
            .HasOne(c => c.User)
            .WithMany()
            .HasForeignKey(c => c.UserId);

        builder.Entity<TaskAttachment>()
            .HasOne(ta => ta.Task)
            .WithMany(t => t.TaskAttachments)
            .HasForeignKey(ta => ta.TaskId);

        // Додавання початкових даних (Seeds) для тестування
        // builder.Entity<IdentityRole>().HasData(new IdentityRole { Name = "Admin",
NormalizedName = "ADMIN" });
        // builder.Entity<IdentityRole>().HasData(new IdentityRole { Name = "Manager",
NormalizedName = "MANAGER" });
        // builder.Entity<IdentityRole>().HasData(new IdentityRole { Name = "Worker",
NormalizedName = "WORKER" });
    }
}
```