КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота

на здобуття освітнього рівня бакалавра

за спеціальністю 121 Інженерія програмного забезпечення

на тему:

РОЗРОБКА КОРПОРАТИВНОГО ВЕБ-СЕРВІСУ ДЛЯ АНАЛІЗУ ТА АВТОМАТИЗАЦІЇ ПРОЦЕСУ ЗАМОВЛЕННЯ ЇЖІ

Виконала студентка 4-го курсу
Майя ЧУДІНОВА

_____
(підпис)

Науковий керівник:

к.т.н., доцент кафедри інтелектуальних програмних систем
Євген ДЕМКІВСЬКИЙ

_____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри інтелектуальних програмних систем
«__»_____2021р.,
протокол №
Завідувач кафедри

Олександр ПРОВОТАР _____
(підпис)

Київ – 2021

РЕФЕРАТ

Обсяг роботи 52 сторінки, 31 ілюстрація, 4 таблиці, 10 джерел посилань.

ВЕБ-СЕРВІС, ДОМЕН, ПРЕДМЕТНО-ОРІЄНТОВАНЕ ПРОЕКТУВАННЯ, ОДНОСТОРІНКОВА АРХІТЕКТУРА, ONION-АРХІТЕКТУРА, ДОМЕННИЙ АНАЛІЗ, ДОМЕННИЙ КОНТЕКСТ, КОМПОНЕНТ, VALUE OBJECT, ENTITY, SPA.

Об’єктом роботи є процес створення веб-сервісу для спрощення процесу замовлення їжі в компаніях. Предметом роботи є прикладний програмний засіб для автоматизації та цифровізації тих чи інших дій користувача на основі його ролі в системі.

Метою роботи є створення програмного засобу автоматизації процесу замовлення їжі для корпоративного використання.

Інструменти розроблення: безкоштовне, вільно поширюване інтегроване середовище розробки Microsoft Visual Studio 2019, Visual Studio Code, мови програмування C#, TypeScript, фреймворки ASP.NET Core, .NET Core, Angular 9, СУБД MariaDb, фреймворк об’єктно-реляційного відображення EntityFrameworkCore, бібліотеки для тестування xUnit, Moq, nuget-пакет для надсилання електронних листів SendGrid.

Результати роботи: виконано загальний огляд архітектури та теорії проектування веб-додатків, розроблено програмний веб-орієнтований продукт для автоматизації процесу замовлення їжі в компаніях (для корпоративного використання), який дозволяє спростити процес замовлення їжі офісними працівниками, проаналізовано можливі шляхи розвитку додатку.

ЗМІСТ

ВСТУП 5

РОЗДІЛ 1. АНАЛІЗ РОЛЕЙ КОРИСТУВАЧІВ СЕРВІСУ ТА ВИЗНАЧЕННЯ ОСНОВНИХ ФУНКЦІОНАЛЬНИХ МОЖЛИВОСТЕЙ ВІДПОВІДНО ДО РОЛІ 9

1.1. Основні функціональні можливості ролі «Працівник» 10

1.1.1. Можливості працівника в системі 10

1.1.2. Переваги, що отримує працівник внаслідок використання системи 10

1.1.3. Автоматизація дій працівника 11

1.2. Основні функціональні можливості ролі «Постачальник» 11

1.2.1. Можливості постачальника в системі 11

1.2.2. Переваги, що отримує постачальник внаслідок використання системи 12

1.2.3. Автоматизація дій постачальника 12

1.3. Основні функціональні можливості ролі «Менеджер» 12

1.3.1. Можливості менеджера в системі 13

1.3.2. Переваги, що отримує менеджер внаслідок використання системи 13

1.3.3. Автоматизація дій менеджера 13

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВИМОГ ДО ВЕБ-СЕРВІСУ 14

2.1. Вимоги від кожної ролі користувачів 14

2.1.1. Вимоги виключно від ролі «Працівник» 14

2.1.2. Вимоги виключно від ролі «Постачальник» 16

2.1.3. Вимоги виключно від ролі «Менеджер» 18

2.2. Нефункціональні вимоги 21

РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА РОЗРОБКА КОРПОРАТИВНОГО ВЕБ-СЕРВІСУ ДЛЯ АНАЛІЗУ ТА АВТОМАТИЗАЦІЇ ПРОЦЕСУ ЗАМОВЛЕННЯ ЇЖІ 24

3.1. Проектування бази даних та ER-модель 24

3.2. Проектування серверної частини веб-сервісу 27

3.3. Проектування клієнтської частини веб-сервісу 35

3.4. Вибір моделі процесу розробки 39

3.5. Аналіз ризиків 40

3.6. Тестування 41

РОЗДІЛ 4. ДЕМОНСТРАЦІЯ ІНТЕРФЕЙСУ 42

4.1. Загальні інтерфейси 42

4.2. Інтерфейси постачальника 43

4.3. Інтерфейси менеджера 44

4.4. Інтерфейси працівника 46

ВИСНОВКИ 48

ДЖЕРЕЛА ПОСИЛАННЯ 49

ДОДАТОК А. ЗАГАЛЬНІ ІНТЕРФЕЙСИ 50

ДОДАТОК Б. ІНТЕРФЕЙСИ ПОСТАЧАЛЬНИКА 50

ДОДАТОК В. ІНТЕРФЕЙСИ МЕНЕДЖЕРА 51

ДОДАТОК Г. ІНТЕРФЕЙСИ ПРАЦІВНИКА 52

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

UX – User experience, користувацький досвід

IDE – Integrated Development Environment, інтегроване середовище розробки

SPA – Single Page Application, односторінковий додаток

MPA – Multi Page Application, багатосторінковий додаток

AJAX – Asynchronous Javascript and XML, асинхронний Javascript та XML

JSON – JavaScript Object Notation

ВСТУП

Оцінка сучасного стану об’єкта розробки. Наразі існує багато різноманітних сервісів, що полегшують процеси замовлення та доставки їжі, наприклад: Glovo, Raketa, Bolt Food, а також сервіси доставки окремих закладів/мереж закладів: Dominos, SushiMaster, TainaBox тощо. Першу групу об’єднує насамперед широке розмаїття постачальників (закладів), що надають послуги з приготування їжі, яку потім доставить окремому кінцевому клієнту кур’єр. Другу групу об’єднує здебільшого те, що разові замовлення доставляються, так само як за допомогою сервісів першої групи, окремому замовнику. Також, очевидно, що асортимент страв в таких сервісах обмежено лише асортиментом певного закладу/мережі закладів, на відміну від додатків першої групи, які надають змогу замовити звідки завгодно в межах досяжних зон доставки.

Загалом, досить багато програмних рішень дозволяють розв’язати проблему доставки їжі. Проте, в більшості таких рішень наявні істотні недоліки, якщо розглядати проблему з точки зору офісних працівників, яких багато, та, які часто замовляють обіди разом. З цих недоліків і виникає можливість створити гнучку систему для корпоративних замовлень, що не пов’язувала б при цьому цілий офіс з одним-єдиним постачальником їжі.

Актуальність роботи та підстави для її виконання. Для офісних працівників наразі не існує доступного сервісу, аби зручно зробити замовлення разом, скажімо, всією командою. Доведеться обраховувати рахунок для кожного працівника власноруч, якщо замовляє в тому ж Glovo чи Raketa хтось один, або ж доведеться телефонувати менеджеру Dominos аби оформити велике замовлення для офісу, яке заразом ще б прийшло одночасно, а не частинами, та при цьому все гаряче і свіже. В другому випадку також наявна наступна проблема: що робити, якщо не вся команда хоче піцу з Dominos? Можливо, половина захоче звідти, половина з іншого місця – це ще досить зручно. Але, що робити, коли команди дві, наприклад, на постійній

основі працюють в одному офісі на різних поверхах? Цілком можливо, що їм не досить зручно взаємодіяти один з одним до обіднього часу, аби узгоджувати його проведення та замовлення, які б влаштували всіх. Тоді обідня перерва займала б вдвічі більше часу, допоки всі будуть задоволені. Було б зручніше якось групувати всі замовлення офісу, навіть якщо команди обідають не разом. Саме з цих міркувань і маємо ідею створення зручного додатку для корпоративного застосування, який полегшував би життя не лише працівникам, а й офісним менеджерам, що організовують обіди для працівників та ведуть їхні рахунки, якщо того вимагає компанія.

Призначення системи полягає в зручній організації корпоративних бізнес-ланчів. Користувачі мають змогу займати різні ролі у системі і, в залежності від них, замовляти ланчі чи надавати послуги постачальника ланчів. Мета системи полягає в частковій автоматизації замовлень їжі корпоративними працівниками та налагодженні зручної взаємодії між компаніями-постачальниками та компаніями-клієнтами за допомогою угод.

Більшість існуючих сервісів являють собою або доставку окремої мережі ресторанів, або кур’єрську доставку на одного замовника.

Відмінність нашої системи полягатиме саме у організації взаємодії між компаніями, а процес замовлення ланчів працівниками має бути легко автоматизований за допомогою шаблонів та автогрупування замовлень.

Мета й завдання роботи. Метою кваліфікаційної роботи є створення програмного засобу для аналізу та автоматизації процесу замовлення їжі офісними працівниками в межах їхніх компаній. Для досягнення цієї мети поставлено наступні завдання:

- Визначити ключові особливості, що демонструватимуть користь додатку з-поміж інших подібних засобів.
- Проаналізувати можливі ролі користувачів програмного засобу, які б створювали зручність використання системи кожним окремим користувачем та дозволяли чітко розділити дії користувачів системи.
- Визначити особливості кожної ролі в системі.

- Сформулювати вимоги до системи: функціональні з боку кожної ролі, функціональні з боку попарної взаємодії ролей, нефункціональні.
- Спроектувати та розробити систему, що відповідає визначеним вище вимогам.
- Окреслити шляхи подальшого розвитку системи.

Об'єкт і методи розроблення. Об’єктом розроблення програмного засобу для аналізу та автоматизації процесу замовлення їжі в компаніях є процес аналізу вимог до програмного засобу, проектування та створення даного засобу згідно з визначеними вимогами. Обрано наступні інструменти створення засобу: IDE Visual Studio 2019, IDE Visual Studio Code, HeidiSQL та СУБД MariaDb, мови програмування C#, TypeScript, фреймворки ASP.NET Core, EntityFrameworkCore, Angular 9, xUnit, Moq.

Можливі сфери застосування. Програмний продукт для аналізу та автоматизації процесу замовлення їжі в компаніях може застосовуватися офісними працівниками для сумісного з іншими працівниками замовлення їжі до офісу, менеджерами компаній для ведення статистики та найпростішого аналізу замовлень працівників.

РОЗДІЛ 1. АНАЛІЗ РОЛЕЙ КОРИСТУВАЧІВ СЕРВІСУ ТА ВИЗНАЧЕННЯ ОСНОВНИХ ФУНКЦІОНАЛЬНИХ МОЖЛИВОСТЕЙ ВІДПОВІДНО ДО РОЛІ

В даному розділі буде проведено аналіз ролей користувачів сервісу з метою подальшого забезпечення якнайзручнішого використання сервісу користувачами будь-якої з визначених ролей. В першу чергу, визначимо та обґрунтуємо ролі користувачів у системі. Природнім чином означається роль працівника (надалі – Працівник) як користувача, який замовляє страви. Дана категорія користувачів буде найпоширенішою.

Також доцільним є виокремлення ролі постачальника (надалі – Постачальник) як користувача, що пропонуватиме на майданчику сервісу страви для подальшого їх замовлення користувачами ролі «Працівник».

Якщо Працівники та Постачальники будуть організовувати свою взаємодію безпосередньо, то система буде такого ж типу, як і Glovo, Raketa, Bolt Food тощо. В цих системах розробники зазвичай поділяють користувачів на ролі не в межах однієї системи, а створюючи цілком іншу систему, наприклад: для замовників – Glovo, для кур’єрів – Glover, для замовників – Raketa (Rocket), а для кур’єрів – Raketa (Rocket) кур’єр, Bolt Food та Bolt Food courier для Bolt відповідно. Ми маємо на меті створення сервісу доставки нового типу, який передбачав би зручність саме корпоративних замовлень.

З цих міркувань, введемо третю роль – менеджер офісу (надалі – Менеджер). Користувачі даної ролі слугуватимуть з деякого боку своєрідними посередниками між Працівниками та Постачальниками. Менеджер офісу необхідний для того, щоб організовано об’єднувати працівників у команди/офіси, що замовляють за однією і тією ж адресою, та налагоджувати зв’язки між своєю компанією та компаніями-постачальниками, що зареєстровані у сервісі. Таким чином, Менеджер слідкує насамперед за тим, щоб всі працівники були правильно згруповані для подальшої роботи сервісу,

а також за тим, що замовляють працівники та на яку суму, аби надалі подбати про сплату.

Важливим моментом є те, що сервіс планується як корпоративний, а отже, ролі користувачів в ньому будуть чітко визначені від початку та незмінні протягом використання сервісу тим чи іншим користувачем. Кожен користувач матиме рівно одну роль у системі, яка визначатиме його можливості у сервісі.

Незалежно від ролі, кожен користувач повинен належати до рівно однієї компанії (чи то замовника, чи то постачальника) та мати змогу редагувати інформацію про себе (окрім робочої електронної поштової скриньки).

Визначимо основні можливості користувачів кожної ролі.

1.1. Основні функціональні можливості ролі «Працівник»

В цьому підрозділі представлено опис та оглядовий аналіз можливостей Працівника в системі.

1.1.1. Можливості працівника в системі

Працівник повинен мати змогу приєднатися до своєї компанії за запрошенням менеджера, переглядати доступні йому страви на майданчику, додавати страви до кошика, формувати замовлення, яке згодом підтвердить Менеджер даного Працівника, або ж зберігати без замовлення певні страви як шаблон для подальших замовлень. Також система повинна надавати Працівнику функціонал отримання звітності по всім замовленням даного Працівника.

1.1.2. Переваги, що отримує працівник внаслідок використання системи

Працівник може замовляти будь-які страви, які забажає і які дозволить йому з певних міркувань та обмежень система, та не піклуватися про групування і погодження свого замовлення з іншими працівниками, що

замовляють обід разом з даним працівником. Система автоматично згрупує замовлення працівника з іншими працівниками оптимальним чином, надішле згруповане замовлення менеджеру працівників, який перегляне його, перевірить, чи все добре, та надішле постачальникові. Працівник зможе відстежувати стан замовлення в додатку, стан замовлення буде оновлюватись постачальником.

1.1.3. Автоматизація дій працівника

Система надасть Працівникові змогу автоматизувати ведення звітності за його витратами на обіди в офісі, а також дозволить зручним чином зберігати улюблені варіанти обідів у шаблони. Сервіс також автоматично згрупує замовлення всіх працівників однієї локації оптимальним чином у одне або декілька замовлень, з якими надалі працюватиме менеджер.

1.2. Основні функціональні можливості ролі «Постачальник»

В цьому підрозділі представлено опис та оглядовий аналіз можливостей Постачальника в системі.

1.2.1. Можливості постачальника в системі

Постачальник повинен мати змогу зареєструватися та зареєструвати в системі компанію, яку він представляє, додавати від імені цієї компанії різноманітні меню та страви, які потім побачать менеджери та працівники. Постачальник повинен мати можливість переглядати пропозиції співпраці від менеджерів компаній-замовників, замовлення груп працівників. Також за допомогою створюваного веб-сервісу повинен мати змогу керувати станом замовлень та приймати чи відхиляти пропозиції співпраці від менеджерів компаній-замовників.

1.2.2. Переваги, що отримує постачальник внаслідок використання системи

Зменшення кількості кур’єрів, що прямують до одного і того ж офісу в ту ж саму обідню перерву, оскільки замовлення групуватимуться системою таким чином, щоб якомога більше індивідуальних замовлень потрапили в одне велике, яке може бути реалізоване як одна доставка.

Ще однією перевагою постачальників в нашій системі є своєрідна їх реклама серед всього офісу, адже, коли менеджер надсилатиме запит на співпрацю постачальникові, і останній його прийме, весь офіс бачитиме меню даного постачальника та зможе в будь-який обід замовити собі декілька його страв. Цілком можливо, що, таким чином, офісні працівники вподобають якогось постачальника та потім замовлятимуть в нього і за межами офісу. В перспективі розвитку даного проекту також можна передбачити більш розгорнуту інформаційну сторінку для компаній-постачальників, де їх представники могли б залишити посилання на засоби індивідуальної доставки страв з цієї компанії, наприклад, посилання на страви компанії у Glovo або сайт компанії, якщо такий наявний.

1.2.3. Автоматизація дій постачальника

Сервіс допоможе постачальнику у веденні звітності за замовленнями до його компанії. В перспективі розвитку системи, можна також передбачити можливість її розширення для створення детальної звітності з різноманітними фільтрами, що спростило б фахівцям компаній-постачальників аналіз.

1.3. Основні функціональні можливості ролі «Менеджер»

В цьому підрозділі представлено опис та оглядовий аналіз можливостей Менеджера в системі.

1.3.1. Можливості менеджера в системі

Менеджер повинен мати змогу зареєструватися та зареєструвати в системі компанію, яку він представляє, або приєднатися до своєї компанії за запрошенням іншого менеджера. Менеджер повинен мати можливість групувати працівників свого офісу у групи, зручні для формування колективних замовлень (зазвичай, за адресою офісу). Менеджер також має бути спроможним переглядати та надсилати постачальникам згруповані замовлення працівників своїх груп, налаштовувати час обіду для групи працівників. Менеджер повинен мати змогу переглядати майданчик всіх зареєстрованих постачальників та пропонувати їм співпрацю. Таким чином, працівники компанії менеджера зможуть замовляти страви у постачальника, якщо між ним та їхньою компанією за допомогою менеджера в системі було укладено угоду про співпрацю.

1.3.2. Переваги, що отримує менеджер внаслідок використання системи

Оцифрування звичних дій менеджера щодо організації обідів у офісі, зменшення кількості дзвінків для замовлення їжі до офісу, можливість отримати звіт за замовленнями своїх працівників, зручна взаємодія з компаніями-постачальниками в корпоративному масштабі: майже все можна зробити навіть без дзвінка – запропонувати співпрацю, дочекатися прийняття або відхилення пропозиції, в разі прийняття всі страви вже доступні працівникам компанії менеджера.

1.3.3. Автоматизація дій менеджера

Система допоможе менеджеру у веденні звітності за замовленнями від працівників його компанії. В перспективі розвитку системи, можна також передбачити можливість її розширення для створення детальної звітності з різноманітними фільтрами, що спростило б аналіз окремих груп працівників, офісів, замовлень за певними кварталами тощо.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВИМОГ ДО ВЕБ-СЕРВІСУ

Розроблювана система призначена для зручного та в деяких аспектах автоматизованого керування обідами офісних працівників, а також зручної організації взаємодії між компаніями-постачальниками та компаніями-замовниками.

Система має концентруватися на мінімізації кількості рутинних дій користувачів в додатку та надавати зручний інтерфейс для взаємодії користувача з системою.

В даному розділі маємо на меті створити конкретні вимоги до веб-сервісу, в першу чергу, користувацькі, як функціональні, так і нефункціональні.

Всі вимоги умовно поділимо на вимоги з боку кожної ролі користувачів, а також вимоги від пар користувачів різних ролей, які б чітко окреслювали зручну взаємодію цих ролей. Вимоги щодо взаємодії всіх трьох ролей між собою одночасно наразі відсутні, адже на даному етапі не передбачається ніяких прецедентів за участю всіх трьох ролей одночасно. Всі процеси проходять поступово між користувачами або однієї ролі, або щонайбільше двох різних ролей одночасно.

2.1. Вимоги від кожної ролі користувачів

2.1.1. Вимоги виключно від ролі «Працівник»

Надалі в усьому підрозділі 2.1.1. актором в системі є Працівник.

Нижче наведено список прецедентів Працівника, що описують його функціональні користувацькі вимоги до системи відповідно до рис. 1.

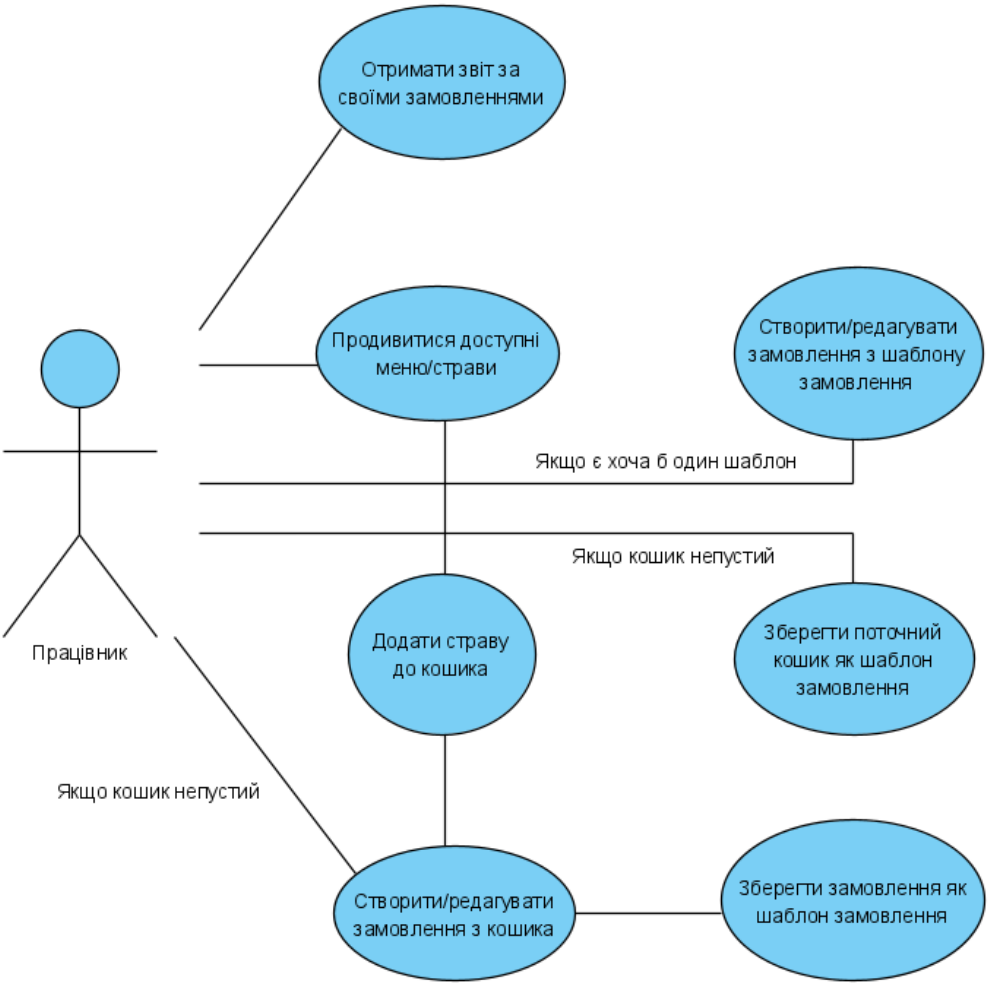

Рис. 1. Use Case діаграма Працівника

1. **ПР1. Перегляд доступних меню та страв**

Працівник може переглянути на майданчику ті меню, що доступні його компанії. Усі страви переглядаються через меню, до якого належать. Передумовою є доступність компанії працівника хоча б одного меню.

2. **ПР2. Додавання страв до кошика**

Працівник може обрати страви з доступних його компанії та додати їх до кошика. Передумовою для додавання страви до кошика є прецедент ПР1: перегляд доступних меню та страв. Після додавання страви до кошика працівник може продовжити перегляд меню або перейти до створення замовлення.

3. **ПР3. Збереження шаблону замовлення з кошика**

Працівник може використати непорожній кошик для збереження списку страв (без створення замовлення) для подальшого використання. Якщо кошик порожній, передумовою є додання до кошика принаймні однієї страви (прецедент ПР2).

4. **ПР4. Збереження шаблону замовлення із замовлення**

Працівник може використати щойно створене замовлення для збереження списку страв для подальшого використання. Передумовою є прецедент ПР5.

5. **ПР5. Створення замовлення**

Працівник може створити замовлення з поточного вмісту кошика або з довільного збереженого шаблону. Даний прецедент як найважливіший у можливостях працівника розглянемо докладніше, описавши для нього потік подій.

Передумови: для даного прецеденту має бути виконаний принаймні один з прецедентів: ПР2, ПР3, ПР4.

6. **ПР6. Отримання звіту за замовленнями**

Працівник може отримати автоматично згенерований звіт з інформацією про свої замовлення.

2.1.2. Вимоги виключно від ролі «Постачальник»

Надалі в усьому підрозділі 2.1.2. актором в системі є Постачальник.

Нижче наведено список прецедентів Постачальника, що описують його функціональні користувацькі вимоги до системи відповідно до рис. 2.

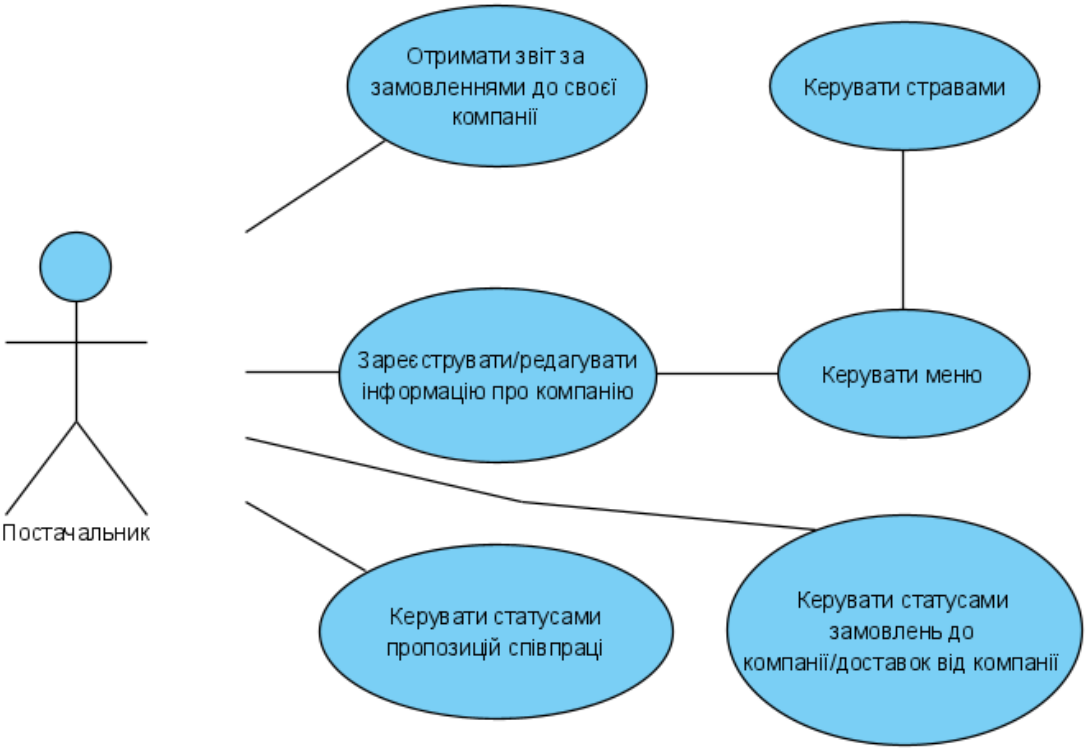

Рис. 2. Use Case діаграма Постачальника

1. **ПО1. Реєстрація компанії/редагування інформації про компанію**

Постачальник може (та має) зареєструвати свою компанію перед початком роботи у системі. Даний прецедент має бути виконано принаймні один раз, він є передумовою до всіх наступних прецедентів постачальника. Редагування інформації доступне після реєстрації компанії в системі в будь-який час.

2. ПО2. Керування меню

Постачальник може переглядати, створювати, редагувати, видаляти свої меню. Передумовою до даного прецеденту є прецедент ПО1.

3. ПО3. Керування стравами

Постачальник може переглядати, створювати, редагувати, видаляти свої страви. Передумовою до даного прецеденту є прецеденти ПО1 та ПО2 (якщо створене хоча б одне меню, частина ПО2: перегляд, інакше частини ПО2: створення та перегляд).

4. ПО4. Отримання звіту за замовленнями до своєї компанії

Постачальник може отримати автоматично згенерований звіт за замовленнями до його компанії за певний період.

5. ПО5. Керування статусами замовлень до компанії/доставок від компанії

Постачальник може редагувати стан замовлення (прийняте, готується, в дорозі, відхилене та ін.)

Постачальник може переглядати замовлення своєї компанії. Інтерфейс має містити список станів замовлення: “Створене”, “Відхилене”, “Підтверджене”, “Доставляється”, “Виконане”. Постачальник обирає поточний стан зі списку. Дана інформація доступна також працівникам, що зробили замовлення.

6. ПО6. Керування статусами пропозицій співпраці

Постачальник може прийняти/відхилити пропозицію менеджера до співпраці за певним меню.Постачальник може переглянути список всіх контрактів, запропонованих йому менеджерами компаній-клієнтів, та прийняти або відхилити ці пропозиції.

2.1.3. Вимоги виключно від ролі «Менеджер»

Надалі в усьому підрозділі 2.1.3. актором в системі є Менеджер.

Нижче наведено список прецедентів Менеджера, що описують його функціональні користувацькі вимоги до системи відповідно до рис. 3.

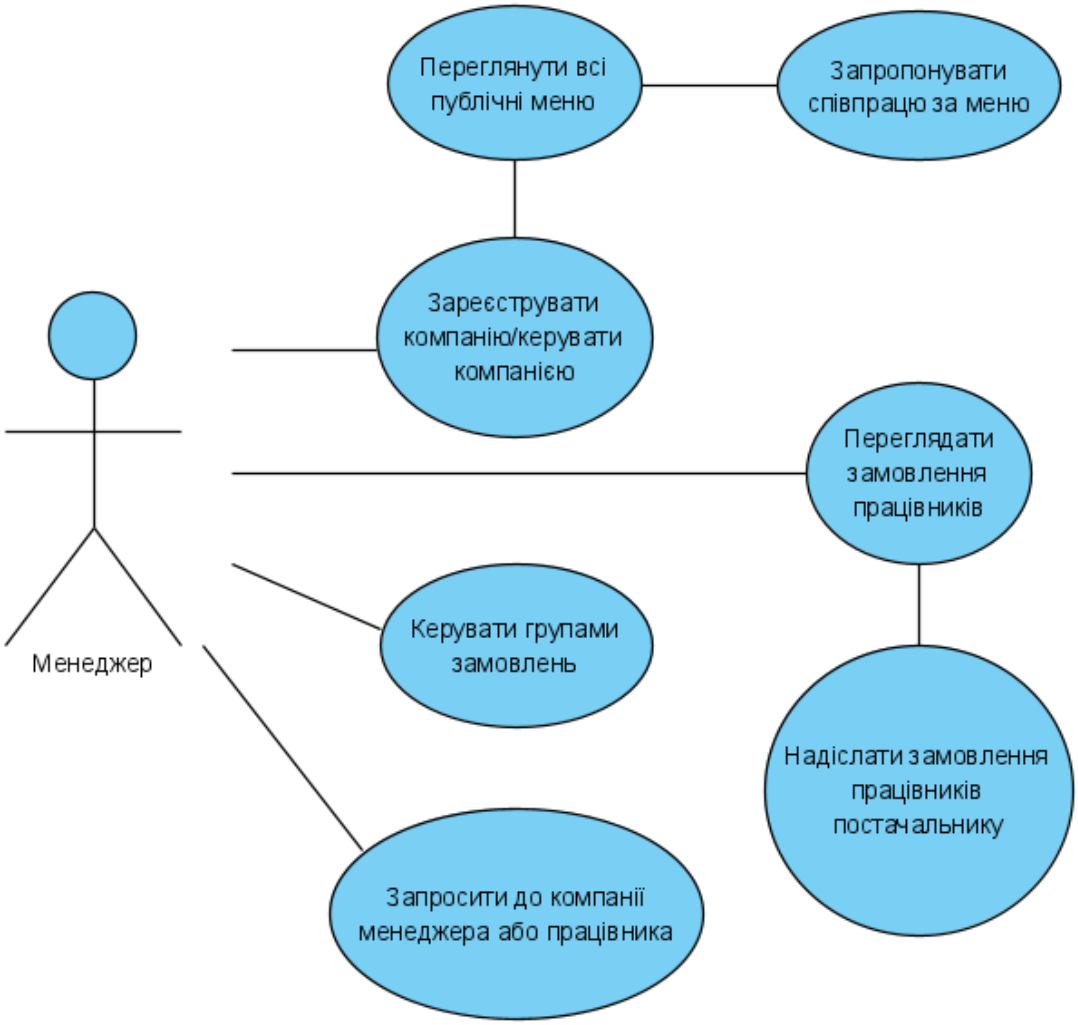

Рис. 3. Use Case діаграма Менеджера

1. **М1. Реєстрація компанії**

Менеджер може (та має) зареєструвати свою компанію перед початком роботи у системі. Даний прецедент має бути виконано принаймні один раз, він є передумовою до всіх наступних прецедентів менеджера. Редагування інформації доступне після реєстрації компанії в системі в будь-який час.

2. **М2. Запрошення до компанії менеджера або працівника**

Менеджер може надіслати запрошення на приєднання до своєї компанії на роль менеджера або працівника.

Після реєстрації своєї компанії, менеджер може переглядати інформацію про неї. Інтерфейс має містити кнопку “Запросити”, яка повинна відкрити форму для заповнення електронної адреси майбутнього працівника або менеджера. Після її заповнення, менеджер може надіслати запрошення на реєстрацію майбутнього працівнику або менеджеру своєї компанії на пошту. В листі має бути наявне посилання, за яким отримувач матиме змогу зареєструватися одразу як працівник або менеджер компанії менеджера, що надіслав запрошення.

3. **М3. Керування групами замовлень**

Менеджер може додавати/видаляти працівника з групи замовлень, редагувати адресу групи, час замовлень. Група замовлень – це концепція, що об’єднує між собою колектив працівників, що замовляють їжу разом, менеджера, що організовує даний колектив та час і періодичність замовлень в цьому колективі.

4. **М4. Пропозиція співпраці.**

Менеджер може запропонувати постачальнику співпрацю за певним меню, тобто домовитись про замовлення працівниками його компанії страв з обраного меню.

Менеджер може переглянути загальний майданчик всіх меню всіх постачальників, подивитися деталі довільного меню та натиснути кнопку “Запропонувати контракт”. Після цього менеджеру має бути надана форма або інформаційне вікно з подробицями запропонованого контракту: назва компанії-постачальника, назва компанії-клієнта, меню, про яке домовляються, та назва (ідентифікатор) контракту. Якщо менеджер підтверджує пропозицію, вона надсилається постачальнику.

5. **М5. Відправлення замовлень**

Менеджер може перевірити замовлення працівників та надіслати їх постачальникам.

6. М6. Отримання звіту за замовленнями

Менеджер може отримати автоматично згенерований звіт з частковою інформацією про замовлення працівників своєї компанії за певний період.

2.2. Нефункціональні вимоги

В таблиці 2.1 представлено опис користувацьких нефункціональних вимог.

Таблиця 2.1

Найменування	Опис
Зручність	Інтерфейс системи має бути зручним, інтуїтивно зрозумілим та не вимагати додаткової підготовки користувачів.
Час відгуку	Час відгуку для звичайних запитів має не перевищувати 10 секунд, для генерування звітів не перевищувати 15 секунд.

В таблиці 2.2 представлено опис некористувацьких нефункціональних вимог.

Таблиця 2.2

Найменування	Опис
Крос-браузерність	Система має працювати у браузерах: Google Chrome, Mozilla Firefox, Microsoft Edge, Opera.
Системні вимоги до сервера	Мінімальні системні вимоги для функціонування серверів системи: Оперативна пам’ять: 4 ГБ Вільне місце на диску: 1 ГБ Тактова частота процесора: 1.6 ГГц

	Кількість ядер процесора: 2 Рекомендовані системні вимоги для функціонування серверів системи: Оперативна пам’ять: 8 ГБ Вільне місце на диску: 1 ГБ Тактова частота процесора: 3 ГГц Кількість ядер процесора: 4

В таблиці 2.3 представлено опис додаткових вимог, що було б бажано задовольнити в перспективі розвитку проекту.

Таблиця 2.3

Найменування	Опис
Доступність	Система має бути доступна 24/7.
Кількість користувачів, що працюють одночасно	Система має обслуговувати щонайменше 10 користувачів, що працюють одночасно та пов’язані спільною базою даних.
Хостинг	Система має бути розгорнута на довільному хостингу за безкоштовним планом, який дозволяє розгортання без використання контейнерів Docker.
Масштабованість	Система має обслуговувати щонайменше 10 користувачів, що працюють одночасно та пов’язані спільною базою даних, та залишати можливість збільшити їх кількість за потреби.

РОЗДІЛ 3. ПРОЕКТУВАННЯ ТА РОЗРОБКА КОРПОРАТИВНОГО ВЕБ-СЕРВІСУ ДЛЯ АНАЛІЗУ ТА АВТОМАТИЗАЦІЇ ПРОЦЕСУ ЗАМОВЛЕННЯ ЇЖІ

3.1. Проектування бази даних та ER-модель

База даних та відповідні зв'язки згенеровані автоматично за рахунок використання Entity Framework Core, тобто сутності бази даних відповідають сутностям, реалізованим у серверній частині додатку.

Проте, код було написано виходячи з певної ER-моделі бази даних (рис. 4). Тобто все ж таки підхід розробки передбачав попереднє чітке планування структури бази даних, яку наведено у вигляді діаграми нижче.

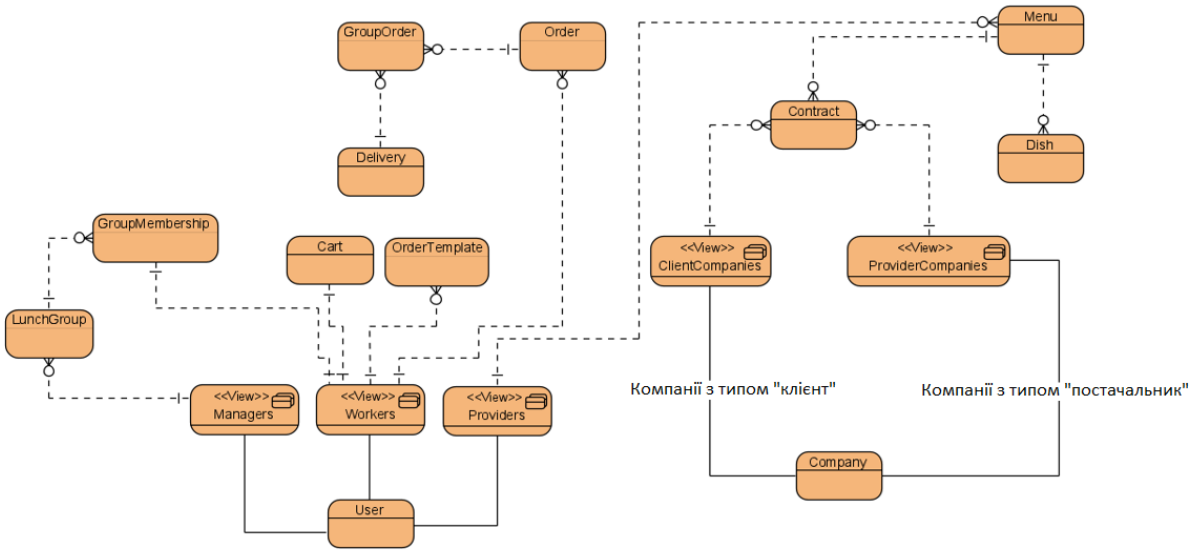

Рис. 4. ER-діаграма системи

Наявні сутності:

Таблиці:

- User

Представляє користувача в системі. Про користувача має бути відома наступна інформація: ідентифікатор, логін, адреса електронної пошти, ідентифікатор компанії, роль, ім'я та прізвище, а також інформація про пароль.

Пароль не має зберігатись у базі даних у прямому вигляді, отже сутність User зберігатиме так звану «сіль пароля» та захешований пароль. Хешування відбуватиметься за допомогою вбудованого у простір імен .NET System.Security.Cryptography алгоритму HMACSHA512.

- Company

Представляє компанію в системі. Про компанію має бути відома наступна інформація: ідентифікатор, назва, опис, тип компанії (компанія, що замовляє їжу, чи компанія, що постачає їжу), поштова адреса компанії (офіційна або уповноваженого представника компанії, адреса офісу компанії (зазвичай компанії розташовані в одній будівлі, дана інформація буде використовуватись лише для опису компанії, але не для логіки).

- Menu

Представляє меню страв в системі. Про меню має бути відома наступна інформація: ідентифікатор, назва меню, інформація про меню (опис), ідетифікатор постачальника, якому належить меню, логічний флаг, що вказує, чи доступне меню зараз, інформація про зображення меню (власне байти зображення або шлях до файлу на сервері).

- Dish

Представляє страву в системі. Про страву має бути відома наступна інформація: ідентифікатор, назва страви, категорія страви (описова інформація про те, чи призначена, наприклад, страва для споживання вегетеріанцями, тощо), склад страви, час приготування, кількість білків, жирів та вуглеводів на 100 г страви, кількість калорій на 100 г страви, ціна, ідентифікатор меню, до якого належить страва, логічний флаг, що вказує, чи доступна страва зараз, інформація про зображення меню (власне байти зображення або шлях до файлу на сервері).

- Contract

Представляє домовленість про співпрацю двох компаній в системі. Про контракт має бути відома наступна інформація: ідентифікатор, номер контракту, ідентифікатор компанії-постачальника, ідентифікатор компанії-

замовника, ідентифікатор меню, про яке узгоджується співпраця, статус домовленості (узгоджена, очікує прийняття компанією-постачальником, очікує підтвердження компанією-замовником).

- Cart

Представляє кошик користувача в системі. Про кошик має бути відома наступна інформація: ідентифікатор, ідентифікатор власника (працівника, адже тільки працівники в системі мають кошик), ідентифікатор поточного постачальника (одне замовлення має бути лише до єдиного постачальника), список страв.

- Order

Представляє замовлення користувача в системі. Про замовлення має бути відома наступна інформація: ідентифікатор, дата доставки, список страв, додаткова інформація, ідентифікатор постачальника, ідентифікатор працівника, ідентифікатор групового замовлення.

- OrderTemplate

Представляє шаблон замовлення користувача в системі. Про шаблон замовлення має бути відома наступна інформація: ідентифікатор, назва, список страв, ідентифікатор постачальника, ідентифікатор працівника.

- LunchGroup

Представляє групу працівників, об'єднаних для замовлень в системі (надалі – групу замовлень). Про групу замовлень має бути відома наступна інформація: ідентифікатор, назва, вихідні, ідентифікатор компанії, логічний флаг, чи є група замовлень активною на даний момент, адреса, ідентифікатор менеджера, що керує групою, дата і час початку поточного періоду замовлень, кінцева дата і час замовлень для працівників, щоб встигнути отримати замовлення в певний день.

- GroupMembership

Представляє сутність зв'язку працівників, менеджера та групи замовлень в системі. Про членство в групі замовленя має бути відома наступна інформація: ідентифікатор члена групи, ідентифікатор групи замовлень.

- GroupOrder

Представляє групове замовлення в системі. Про групове замовлення має бути відома наступна інформація: ідентифікатор, ідентифікатор постачальника, ідентифікатор групи замовлень, дата доставки замовлення.

- Delivery

Представляє доставку в системі. Про доставку має бути відома наступна інформація: ідентифікатор, ідентифікатор групового замовлення, що доставляється, час доставки, поточний статус доставки.

Представлення умовні, з EntityFrameworkCore зручніше використовувати спеціальні фільтри в поєднанні з інтерфейсом IQueryable на стороні сервера, аніж створювати та маніпулювати SQL View у традиційному розумінні. З використанням цих фільтрів можна виділити наступні логічні представлення:

- Workers

Всі користувачі, які мають роль працівника. Підмножина таблиці User.

- Providers

Всі користувачі, які мають роль постачальника. Підмножина таблиці User.

- Managers

Всі користувачі, які мають роль менеджера. Підмножина таблиці User.

- ClientCompanies

Всі компанії, що зареєстровані як ті, що замовляють їжу.

- ProviderCompanies

Всі компанії, що зареєстровані ті, що постачають їжу.

3.2. Проектування серверної частини веб-сервісу

Для створення серверної частини додатку оберемо Onion-архітектуру як одну з реалізацій так званої «чистої» архітектури [5] [6] [8]. Основна ідея такої архітектури в контролі зв’язаності, а саме контролі напрямків залежностей. Головне правило формулюється наступним чином: частини програми можуть

залежати від інших частин тоді і тільки тоді, коли залежні частини знаходяться далі від ядра в архітектурі, аніж ті, від яких вони залежать (рис. 5). В центрі onion-архітектури бізнес-логіка. Вона ж і є логічним ядром програми, яке має бути повністю незалежним від інших частин програми, які, в свою чергу, можуть використовувати ядро [5] [6] [7]. Така архітектура ідеально пасує до об'єктно-орієнтованого підходу в розробці, оскільки в центрі програми опиняються саме класи та об'єкти, не таблиці бази даних та не будь-які інфраструктурні чи сторонні сервіси [7].

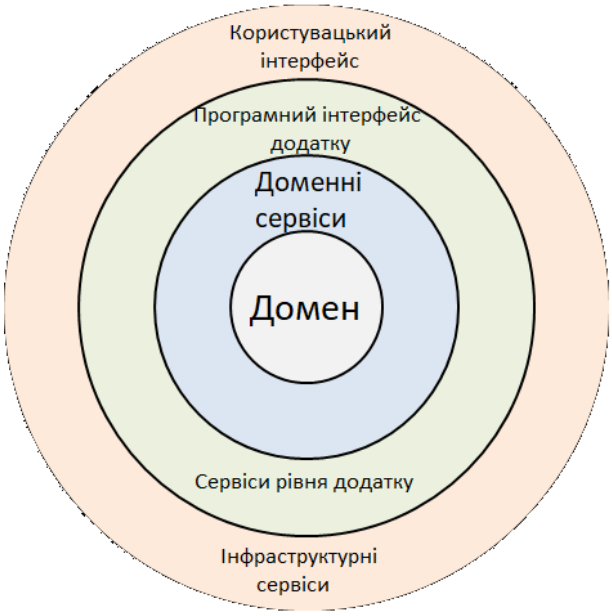

Рис. 5. Схема onion-архітектури

Логічне ядро – бізнес-логіка – програми називається доменом. А ключовим принципом проектування та розробки в цілому в onion-архітектурі відповідно є предметно-орієнтоване проектування (Domain Driven Design). Бізнес-логіка відповідає за керування станом та поведінкою логічних моделей, включаючи валідацію стану та обмеження на поведінкові переходи між станами тощо.

Другим шаром у напрямку від центру ядра програми є сукупність всіх доменних сервісів, що надають функціонал для додаткової маніпуляції доменом, якщо потрібно. Разом з третім шаром, який містить сервіси рівня

додатку, другий надає інтерфейси для взаємодії з доменом для запитів на отримання та збереження сутностей домену. При цьому реалізація даних інтерфейсів в жодному разі не має знаходитися в цих двох шарах.

Таким чином, ядро додатку складає лише домен та інтерфейси взаємодії з доменом, конкретна реалізація яких надається за допомогою механізму впровадження залежностей (dependency injection). Даний механізм заснований на принципі SOLID Dependency Inversion – принципі інверсії залежностей, при якому абстракції не залежать від деталей та реалізації, а модулі як верхнього, так і нижнього рівнів, залежать від абстракцій.

Реалізації інтерфейсів знаходяться на верхньому рівні, заразом з інфраструктурними та сторонніми сервісами і власне механізмами взаємодії з базою даних.

Розглянемо плюси та мінуси вибору onion-архітектури. Серед переваг можна виділити наступні:

1. Швидке та зручне юніт-тестування бізнес логіки, що випливає з абсолютної незалежності ядра програми від інших компонентів програми.

2. Системи, побудовані за парадигмами Onion-архітектури, добре масштабуються за умови достатньо гідної реалізації (мається на увазі проектування згідно з принципами архітектури).

3. Бізнес-логіка системи не залежить від ядра бази даних, тому при створенні міграцій (наприклад, за допомогою EntityFrameworkCore) задача зміни ядра бази даних не створює труднощів. Фактично, окрім переносу даних, потрібно змінити лише один драйвер інфраструктурного рівня, що відповідає за підключення до бази даних. Взаємодія з базою даних залишається при цьому незмінною в коді. Звісно, необхідно зауважити, що при переході на іншу мову запитів до бази даних, коли, наприклад, відбувається перехід з MS SQL Server до MongoDb, даний пункт не має ваги, оскільки така зміна ядра носить концептуальний характер, так як змінюється сам принцип ядра, а не реалізація.

4. Клієнтська частина застосунку залежить від серверної, але не навпаки. Ці дві частини – повністю окремі додатки, що займають на хості два

порти та можуть використовуватись незалежно один від одного (уся доступна «ззовні» частина серверу та частина клієнта, що не вимагає запитів до серверу, наприклад, сторінки помилок та ін.). З усього сказаного в даному пункті випливає, що зміна клієнтської частини додатку також не створює додаткових проблем на стороні сервера, оскільки не потребує його зміни взагалі.

Серед недоліків підходу можна виділити наступні:

1. Для початку розробки та частково в ході реалізації проекту необхідний докладний аналіз системи. Від проектування на початку залежить успіх створення системи, адже починається розробка з створення ядра бізнес-логіки, від якого залежить майже все в проекті. Якщо допустити серйозну помилку у визначенні бізнес-логіки, що призведе до некоректної/небажаної поведінки системи, виправлення знадобиться зробити майже у всіх шарах системи, а саме частинах, що стосуються контексту, в якому допущено помилку.

2. Дана архітектура спрощує складні проекти, але ускладнює проектування досить простих проектів. Тому, слід обирати її, коли бізнес-логіка дійсно може бути заплутаною і необхідно ввести послаблення залежностей у коді програми. В нашому випадку, це суб’єктивний недолік, і я вважаю, що дана архітектура варта впровадження заради майбутньої масштабованості, швидкого тестування а також майже «безкоштовної» зміни ядра бази даних або ж клієнтської частини застосунку.

Враховуючи все вищесказане, можемо зробити висновок, що вибір Onion-архітектури є виправданим та стане в нагоді в подальшому аналізі завдяки своїм методам проектування і розробки (доменний аналіз, предметно-орієнтоване проектування).

Визначимо основні доменні контексти програми та спроектуємо їх у вигляді діаграми доменів (яка, по суті, є діаграмою класів із виділеними доменами). Доменний контекст – це логічно окрема частина програми, яка виконує певну незалежну під-задачу в програмі. Всередині доменних контекстів класи можуть бути зв’язані, оскільки вони формують розв’язок

єдиної задачі, а отже є сильно зв’язними. Також слід ввести поняття сутності (entity) та типу значення, що зазвичай використовують без перекладу та кажуть value-object. Сутність – це клас, що представляє у програмі певний доменний предмет, який має стан та власну ідентичність. Value-object – клас або структура, яка слугує для створення полів класів типу сутності, але при цьому не має власного стану та ідентичності. Рівність value-object’ів визначається сукупністю їх полів, в той час як рівність сутностей визначається рівністю їх ідентифікаторів. Сутностям в базі даних обов’язково відповідають таблиці, в той час як value-object’и зберігаються у базі даних лише як структурні частини сутностей.

Перший доменний контекст – керування користувачами (рис. 6). Даний контекст міститиме рівно одну доменну сутність – користувача – та буде розв’язувати задачу керування користувачами в програмі, а саме: отримання та зберігання даних про користувача, надання сервісів аутентифікації та авторизації.

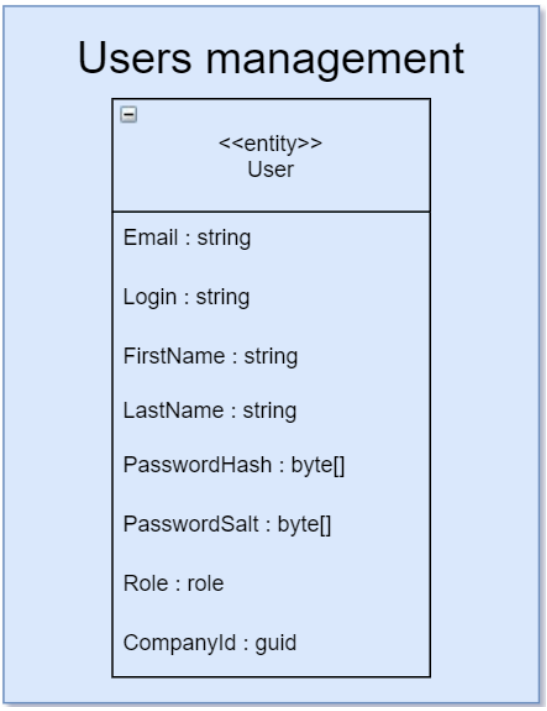

Рис. 6. Діаграма класів контексту керування користувачами

Наступний доменний контекст – контекст компаній (рис. 7). Даний контекст міститиме сутності компаній та контрактів (домовленостей) та буде розв’язувати задачу реєстрації та керування інформацією про компанії, а також створення зв’язків між компаніями-постачальниками та компаніями-клієнтами у вигляді контрактів.

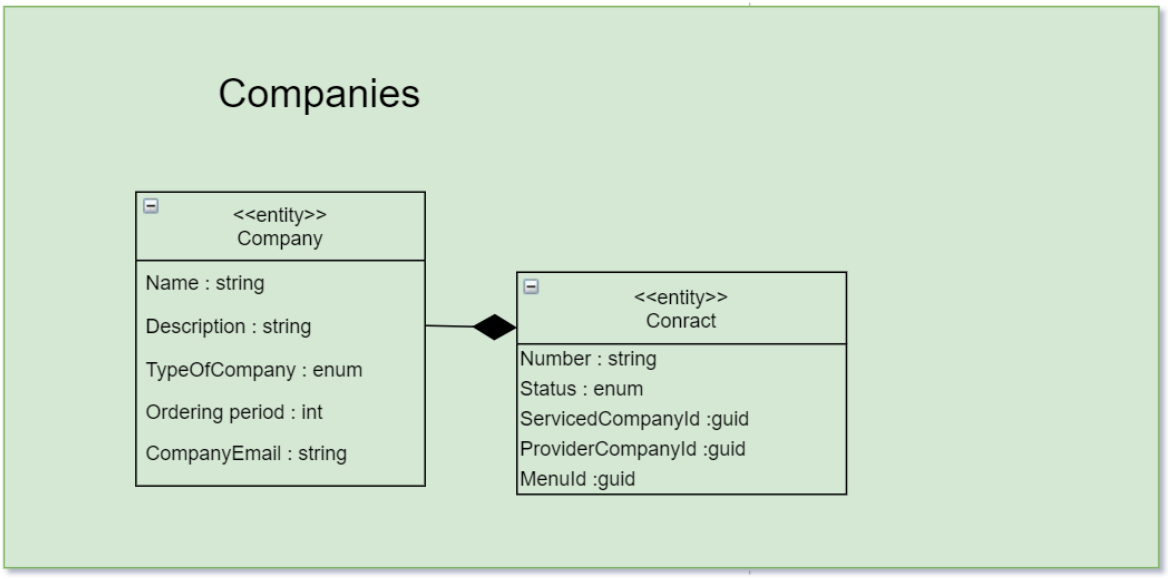

Рис. 7. Діаграма класів контексту компаній

Доменний контекст меню (рис. 8) міститиме сутності для керування меню та стравами меню, а також сервіс для взаємодії користувачів з множиною всіх меню та її підмножинами залежно від ролі користувача разом з допоміжними класами типу value-object.

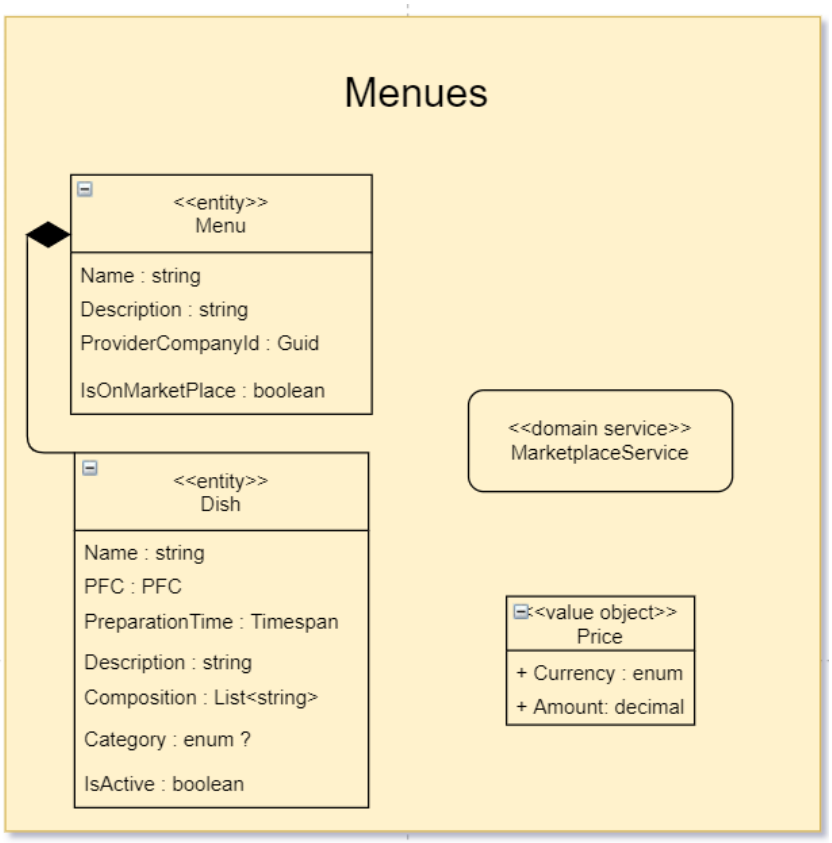

Рис. 8. Діаграма класів контексту меню

Наступні два контексти – групових замовлень (рис. 9) та груп замовлень (рис. 10) – розкривають свою функціональність з використанням запитів до перших трьох контекстів. Проте, вони не залежать від попередньо визначених контекстів у коді, а містять лише ідентифікатори для запитів до відповідних контекстів. Серед сутностей контексту групових замовлень можна виділити наступні: групове замовлення, замовлення, шаблон замовлення та кошик. Всі ці сутності розв’язують задачу створення індивідуальних замовлень працівниками, а сутність групове замовлення за рахунок взаємодії з контекстом груп замовлень формуватиметься із всіх замовлень членів групи (можливо що для однієї групи в той самий обід буде декілька групових замовлень).

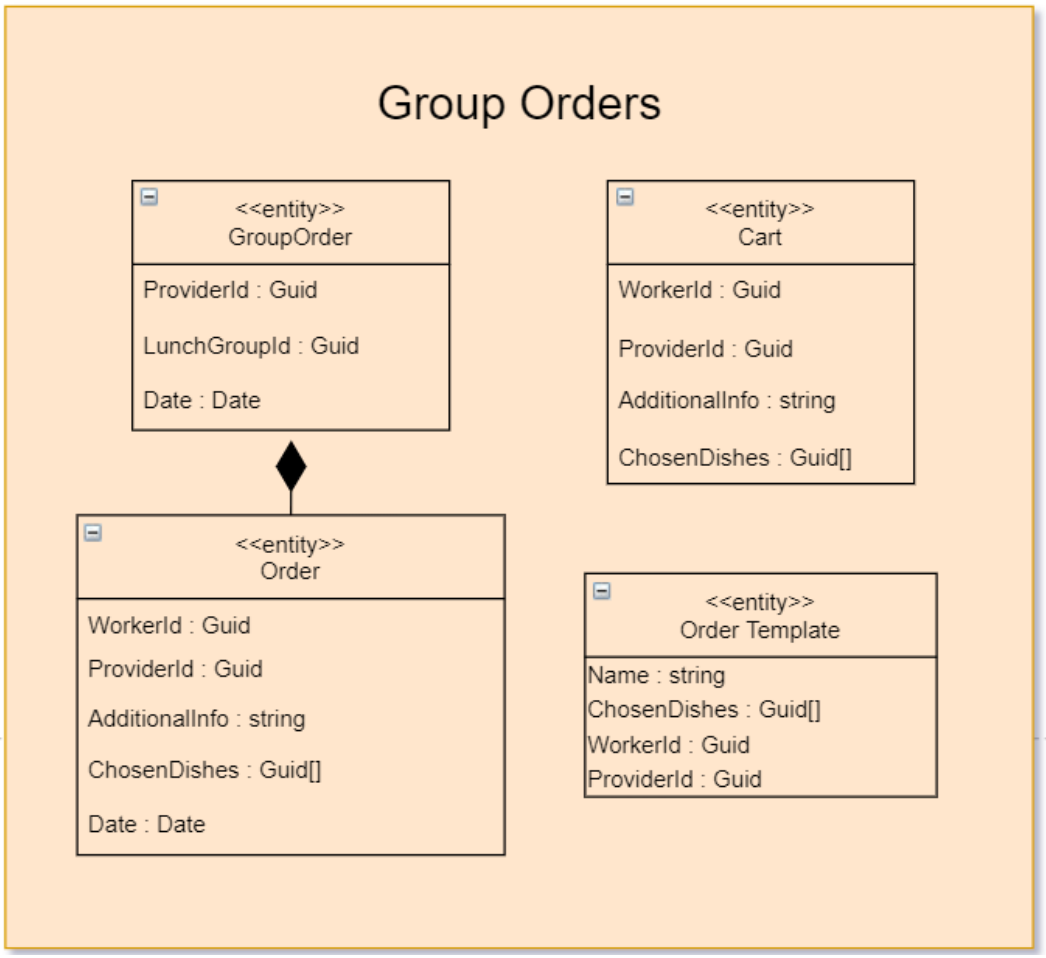

Рис. 9. Діаграма класів контексту групових замовлень

У контексті груп замовлень наявні наступні сутності: група замовлень та доставка, а також клас типу value-object адреса, що зберігатиме повну інформацію про адресу офісу. Ці сутності розв’язують задачу групування працівників у групи та адміністрування цих груп менеджерами. Групи логічно формувати для кожного офісу компанії, проте, якщо необхідні групи для замовлень в той самий офіс, але в різний час доби, доцільно сформувати групу для кожного часу замовлення.

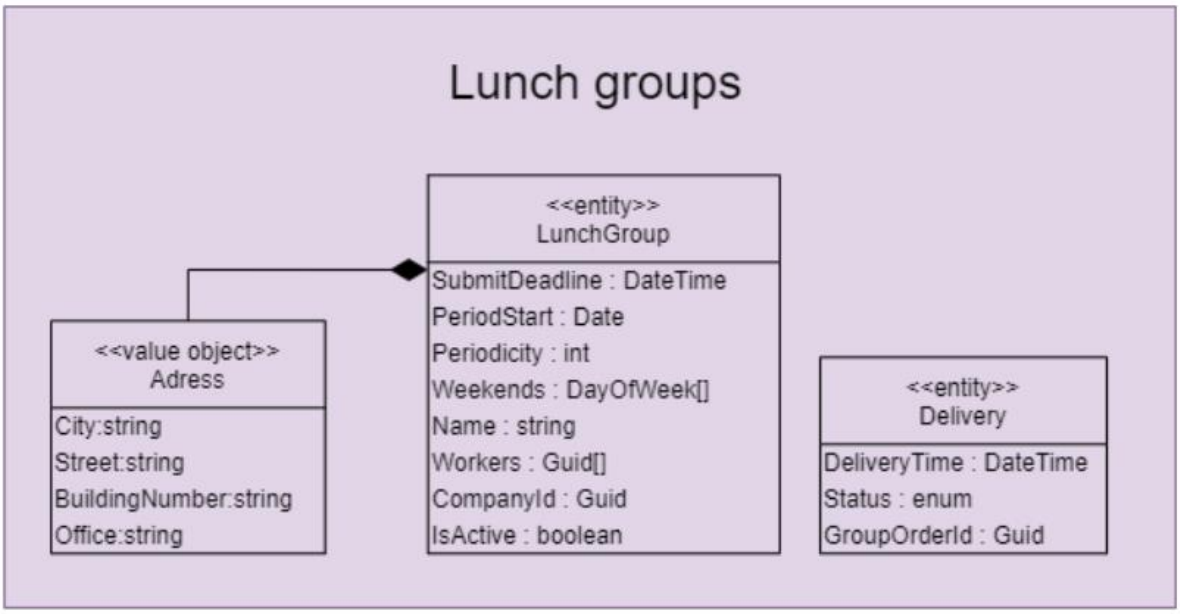

Рис. 10. Діаграма класів контексту груп замовлень

Слід зазначити, що контексти не взаємодіють один з одним безпосередньо, саме тому на діаграмах в класах зберігаються не прямі посилання на об’єкти з іншого контексту, а ідентифікатори, необхідні для отримання інформації про об’єкти. Сама ж реалізація взаємодії між контекстами полягає у створенні шлюзу у вигляді HTTP-клієнта на стороні сервера. Запити з клієнтської частини приходять до нашого умовного шлюзу, який в свою чергу робить всі необхідні запити до контекстів додатку та формує з них відповідь для клієнтської сторони. Це можна було б назвати окремим «керуючим» контекстом програми, адже ця її частина оркеструє всі інші, проте це не контекст, оскільки окрім HTTP-запитів до інших контекстів і сервісу звітів, шлюз не містить доменних сутностей.

3.3. Проектування клієнтської частини веб-сервісу

Архітектура клієнтської частини додатку – Single Page Application (за допомогою Angular). Компоненти (умовні сторінки) загалом відповідають сутностям доменних контекстів (проте є також допоміжні компоненти).

Розглянемо теорію архітектури SPA, а саме відмінності такої архітектури від традиційної багатосторінкової архітектури (MPA – Multi Page Application).

Рис. 11. Традиційна архітектура клієнтської частини

При багатосторінковій архітектурі (рис.11) формується наступна ситуація:

1. Клієнту необхідно відобразити нові дані.

2. Клієнт виконує запит до сервера, щоб отримати цілком нову HTML-сторінку.

3. Сервер надсилає нову сторінку, тепер клієнт має перезавантажити всю сторінку (замінити стару на нову).

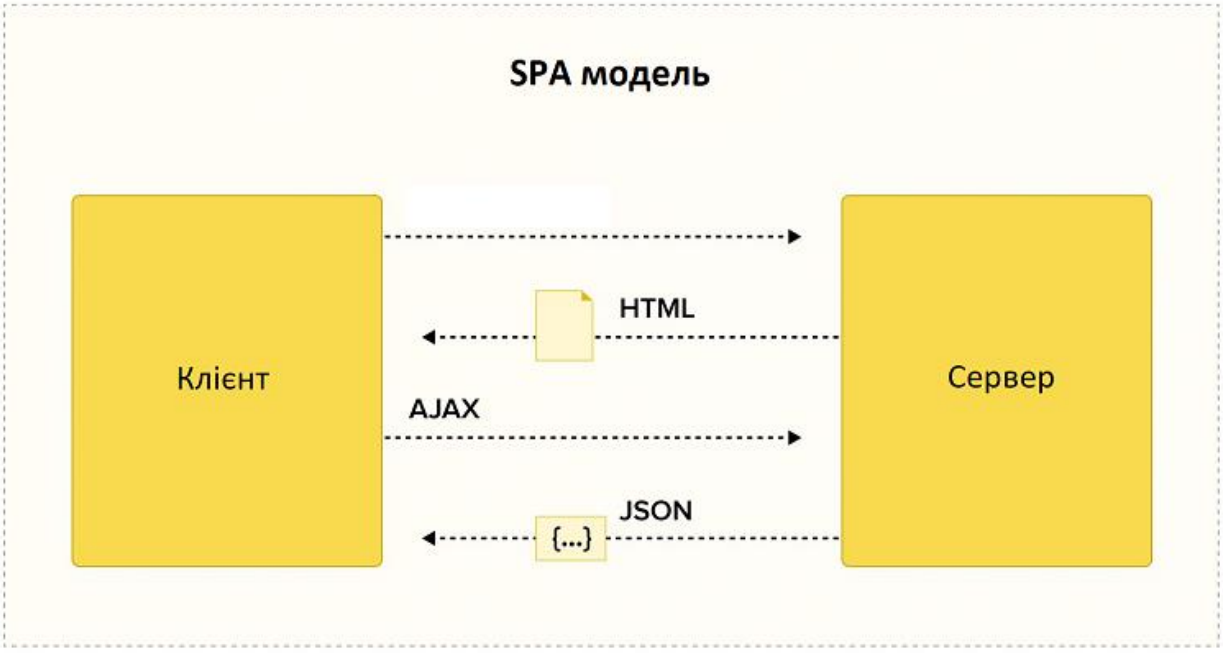

Рис. 12. Односторінкова архітектура клієнтської частини

При односторінковій архітектурі (рис. 12) в даних умовах формується наступна ситуація:

1. Клієнту необхідно відобразити нові дані.

2. Клієнт виконує AJAX-запит до сервера.

3. Сервер надсилає не всю HTML-сторінку, а лише змінені дані (зазвичай у JSON-форматі). Клієнт може перевідобразити лише оновлені дані, без необхідності перезавантажити всю сторінку.

Розглянемо причини вибору саме SPA-архітектури клієнтської частини.

Порівняно з MPA маємо наступні переваги:

- Зручний та швидкий UX

Можливість не перезавантажувати сторінку повністю не лише оптимізує швидкість завантаження «сторінки» для користувача, а ще й робить інтерфейс зручнішим в цілому через відсутність необхідності постійного оновлення сторінки для оновлення даних. Односторінкові додатки також набагато швидші та ефективніші, оскільки призводять до зменшеного потоку даних між клієнтом та сервером, бо запитуються та надсилаються лише змінені дані.

Також швидкість досягається внаслідок того, що рендеринг відбувається на стороні клієнта, а не на стороні сервера, що значно зменшує навантаження на останній.

- Можливість локального кешування

Після першого отримання даних від сервера, клієнт кешує та зберігає локально всі дані, отримані від сервера, а тому використання додатку (з частковою функціональністю) можливе при зникненні підключення до Інтернету. Коли підключення буде відновлено, клієнт синхронізується з сервером для отримання нових даних та відновлення повної функціональності [1] [2].

- Гарна адаптивність для мобільних пристроїв

Під даним пунктом мається на увазі в першу чергу легкий переніс клієнта для використання на мобільних пристроях. В цей же час, такі фреймворки як Angular, React, Vue та деякі інші дозволяють розробляти компоненти одразу і для мобільних і для інших пристроїв, додаючи один чи два параметри до будівельних компонентів сторінки (текстові поля, контейнери, слайдери тощо). А можливість ніколи не перезавантажувати сторінку особливо цінується при доступі до сайту з мобільного пристрою.

- Розділення серверної та клієнтської частини застосунку

Дана перевага призводить до ще однієї, яку можна було б виділити в окрему за наявності відповідної цілі: перевикористання серверного програмного інтерфейсу (API), що значно спрощує переніс та адаптацію клієнтської частини додатку для використання на багатьох типах пристроїв. Також розділення серверної та клієнтської частин прибирає залежність між частиною коду, що маніпулює даними, та частиною, що ці дані відображає. Тому зручно і код відлагоджувати, і знаходити помилки, і, заразом, захищати дані від небажаних дій з боку клієнта.

- Зручне відлагодження за допомогою Chrome Dev Tools у браузері Google Chrome

Дана перевага випливає з відсутності необхідності постійно перезавантажувати сторінку та стає в нагоді, коли потрібно відлагодити код клієнтської частини.

Щодо недоліків у порівнянні з MPA, можна виділити наступні:

- Втрати пам’яті

Слухачі подій, що використовуються при односторінковій архітектурі, можуть спричинити втрату пам’яті на клієнті (memory leak). Втрати пам’яті можуть статися в будь-якому клієнтському додатку, але в даному випадку небезпека криється в тому, що сторінка ніколи не перезавантажуються, накопичуючи можливі помилки [1] [4].

- Безпека

Міжсайтовий скриптинг (cross-site scripting [3]) є причиною більшої вразливості односторінкових додатків.

- Повільне перше завантаження

Через рендеринг на стороні клієнта, а не сервера, перше завантаження додатку може бути повільним, оскільки завантажується вся HTML-сторінка – кістяк клієнта.

- Необхідність підтримки Javascript на клієнті

Через рендеринг на стороні клієнта, Javascript необхідний для функціонування клієнтської частини додатку. Даний недолік не є вагомим при орієнтації на «звичайного» користувача, адже останній не відключає підтримку Javascript у своєму браузері.

Враховуючи описані переваги та недоліки односторінкової архітектури, можемо зробити висновок, що її вибір гармонічно підходить під обрану архітектуру сервера, значно покращує UX та полегшує розробку.

3.4. Вибір моделі процесу розробки

Оберемо каскадну модель процесу розробки, оскільки паралелізм в розробці за умови наявності всього одного розробника неможливий, а

виконання останнім декількох різних по змісту задач одночасно є недоцільним [9].

3.5. Аналіз ризиків

Короткий аналіз ризиків наведено у таблиці 3.1. З базовою теорією управління ризиками при розробці ПЗ ознайомлено за допомогою [10].

Таблиця 3.1

Ризик	**Коментар**	**Імовір-ність**	**Збитки**	**Стратегія вирішення**
Недопланування	Через складність системи, може бути недооцінено мінімальну кількість бізнес-правил, які необхідні для коректного функціонування системи	середня	терпимі	При розробці додавати в оцінений час ще 20% на виправлення
Недостатнє розуміння предметної області	Через відсутність експерта з організації корпоративних ланчів може статися впровадження зайвих бізнес-правил або ухилення від	висока	незначні	Декілька етапів тестування, впродовж яких нові запитання розробникам варто з’ясовувати з третьою стороною (друзями,

	необхідних бізнес-правил			викладачами, власними офіс-менеджерами)

3.6. Тестування

Написано базові юніт тести з використанням бібліотек xUnit та Moq для C#. Дані тести слугують в першу чергу для перевірки при підготуванні нової версії та перевіряють здебільшого коректність викликів до доменних контекстів, оскільки валідація, що знаходиться у ядрі (на доменному рівні), зачіпається лише безпосередньо при зміні коду ядра. Тому для ядра тести не доцільні, оскільки, змінюючи код ядра, ми змінюємо бізнес-правила, які немає сенсу перевіряти старими тестами після змін. Тестові проекти наведено на рис. 13. Присутній чіткий поділ юніт тестів як між контекстами, так і між шарами в межах контексту (шари, очевидно, можуть посилатися один на одного із обмеженнями, визначеними в межах обраної архітектури.

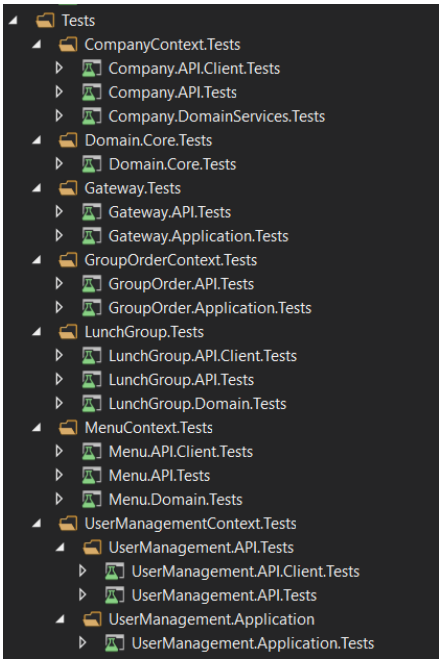

Рис. 13. Проекти з тестами додатку

Мануальне тестування було обмежено незадокументованим тестуванням базових інтерфейсів користувача безвідносно до ролі, інтерфейсів постачальника, менеджера та працівника окремо, а також дослідницьким тестуванням.

Більша частина тестування була виконана одразу після розробки останнього контексту додатку та шлюзу, а також під час розробки кожного окремого контексту.

РОЗДІЛ 4. ІНТЕРФЕЙС КОРИСТУВАЧА ВЕБ-СЕРВІСУ

4.1. Загальні інтерфейси

На рис. 14 наведено базову сторінку входу в систему. На даному етапі користувач може або увійти в систему за допомогою вводу логіну та пароля, або обрати одну з наступних опцій: зареєструватися, натиснути «Забули пароль?».

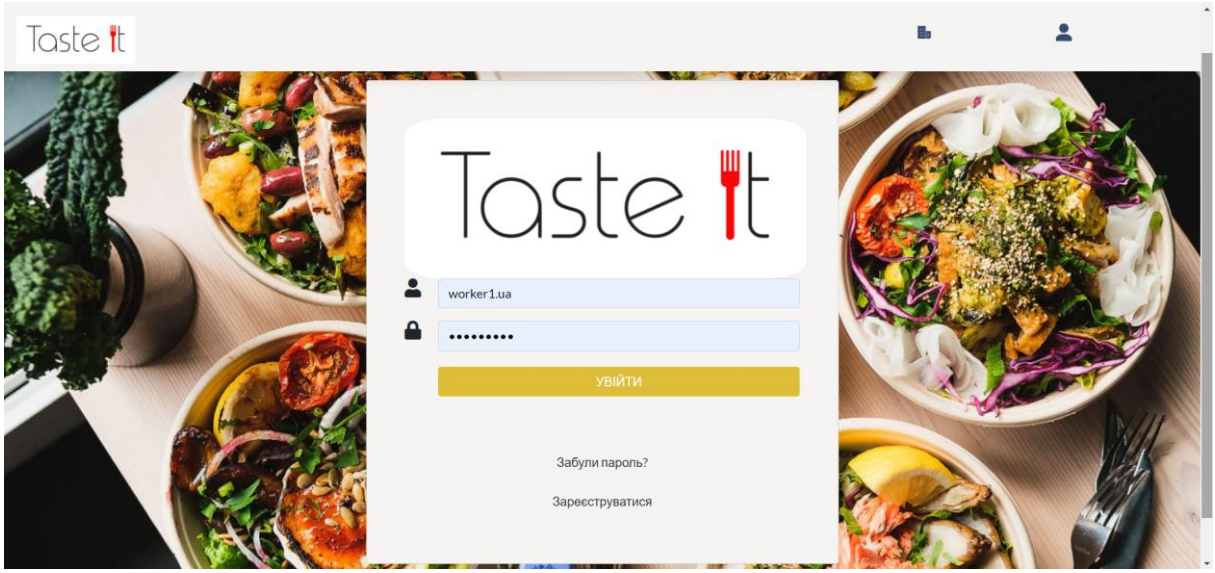

Рис. 14. Сторінка входу в систему

На рис. 15 наведено сторінку перегляду та редагування інформації користувача.

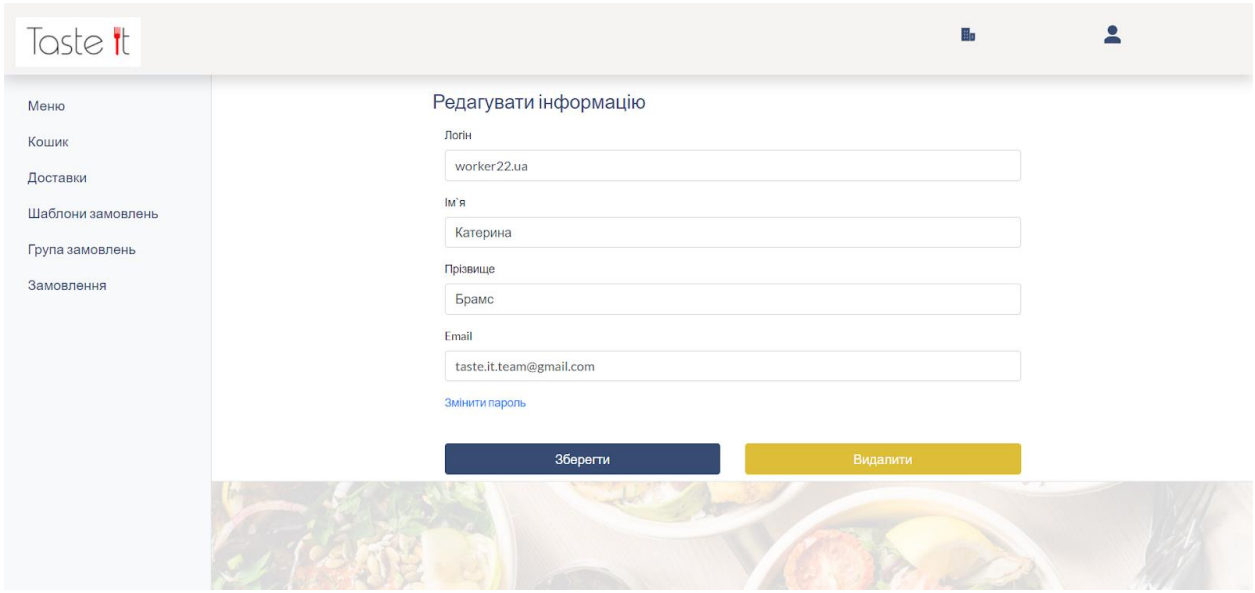

Рис. 15. Сторінка інформації про користувача

Більш докладно ознайомитися з загальними інтерфейсами можна в додатку А.

4.2. Інтерфейси постачальника

На рис. 16 представлено сторінку меню постачальника. На даній сторінці постачальник має змогу додати/видалити певні меню з майданчика (щоб їх поки не бачили інші користувачі, наприклад, коли меню з технічних причин зараз недоступне).

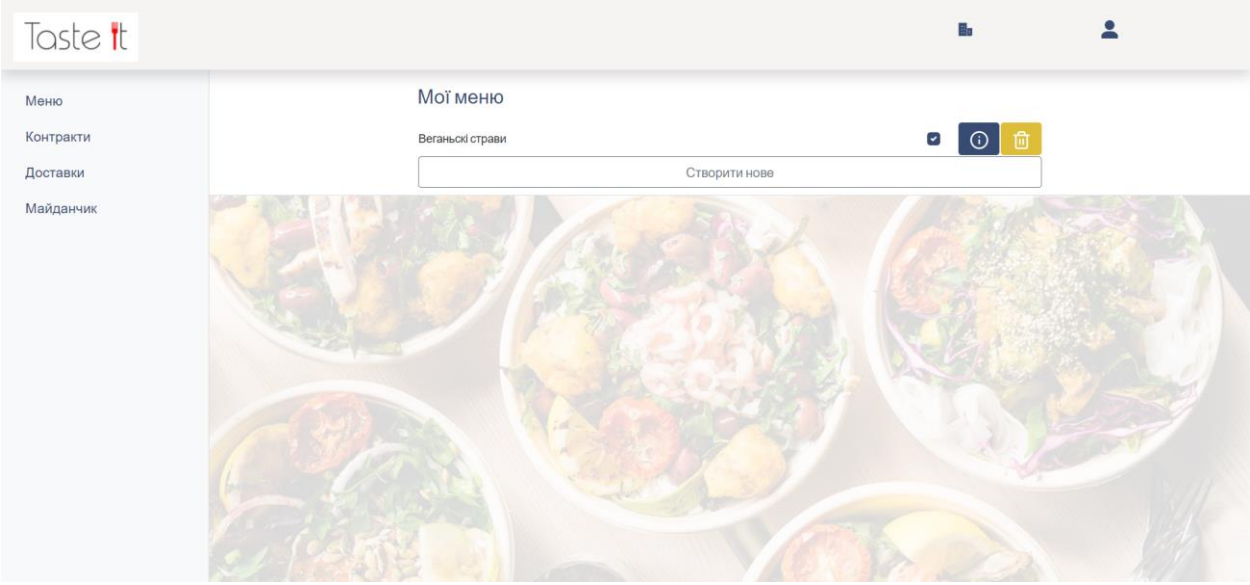

Рис. 16. Сторінка меню постачальника

На рис. 17, рис. 18 представлено сторінку для перегляду/редагування детальної інформації про меню, в тому числі переходу до сторінок детальної інформації про страви.

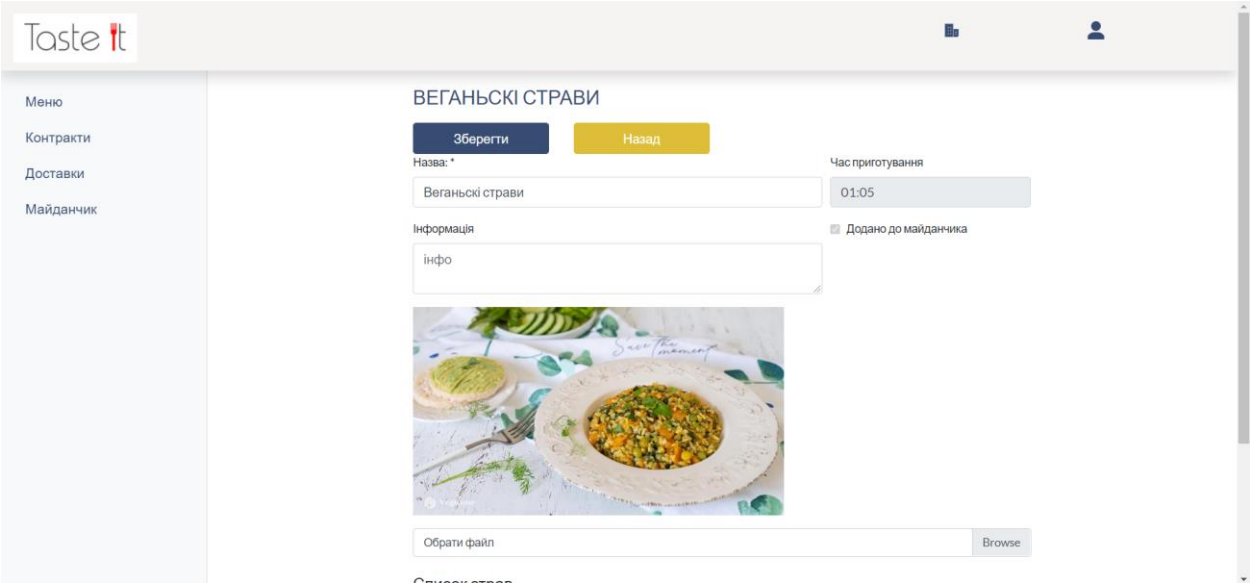

Рис. 17. Сторінка детальної інформації про меню

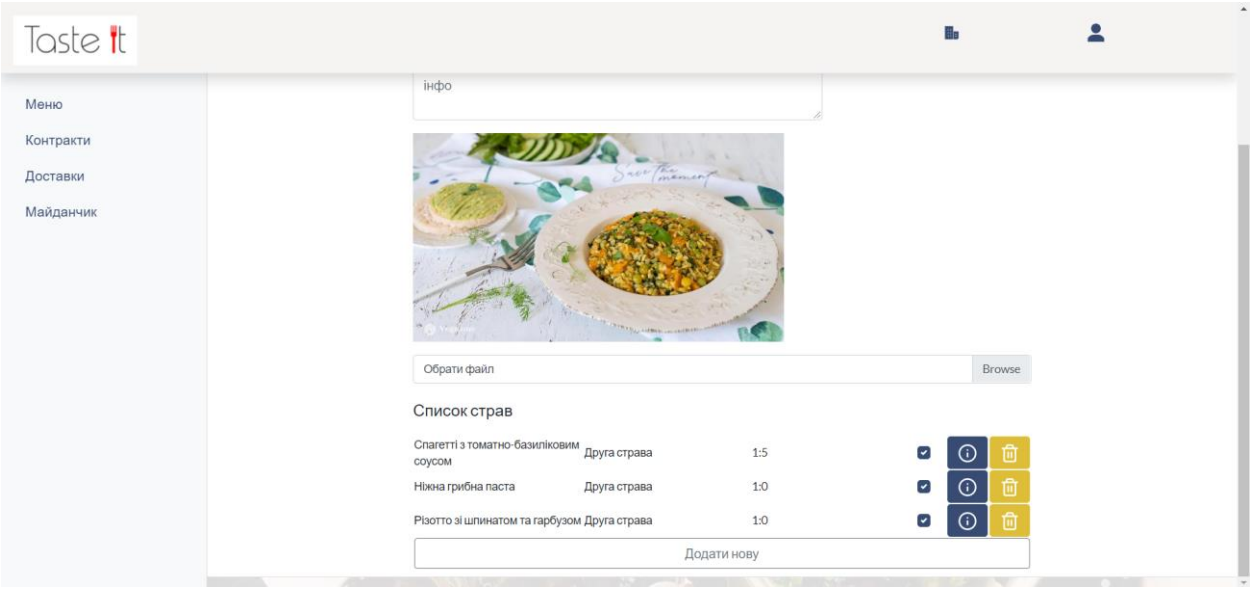

Рис. 18. Сторінка детальної інформації про меню (продовження)

Більш докладно ознайомитися з інтерфейсами постачальника можна в додатку Б.

4.3. Інтерфейси менеджера

На рис. 19 представлено майданчик всіх доступних меню. Даний інтерфейс також доступний постачальнику, проте, в основному, ним користується саме менеджер.

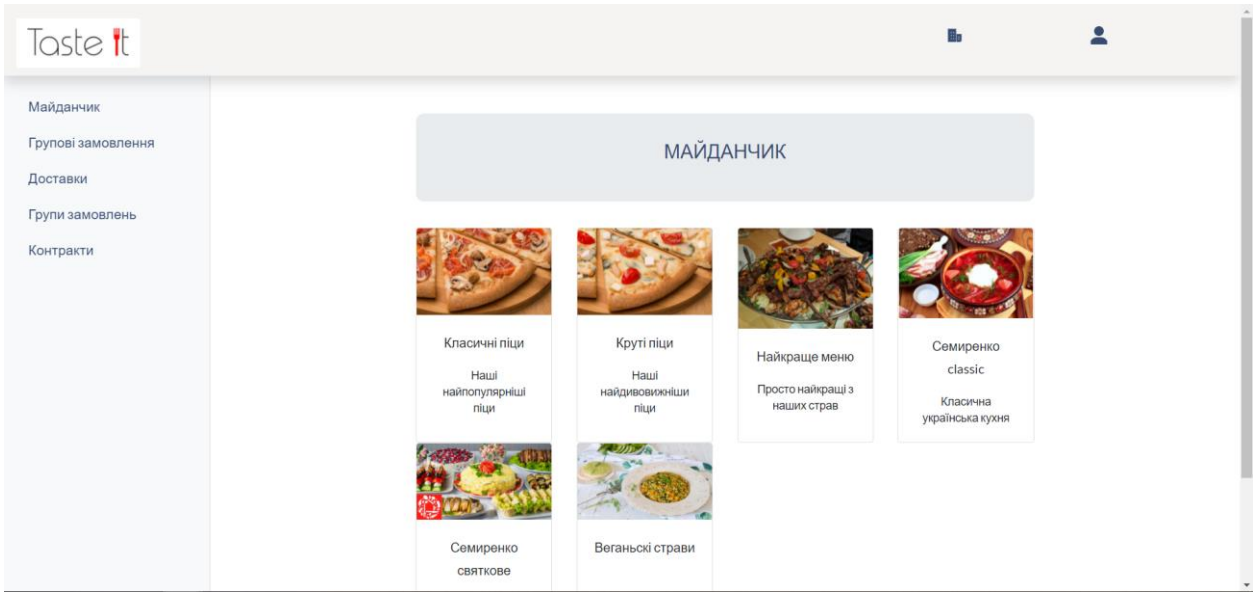

Рис. 19. Сторінка з усіма активними меню

На рис. 20 наведено сторінку для перегляду групових замовлень.

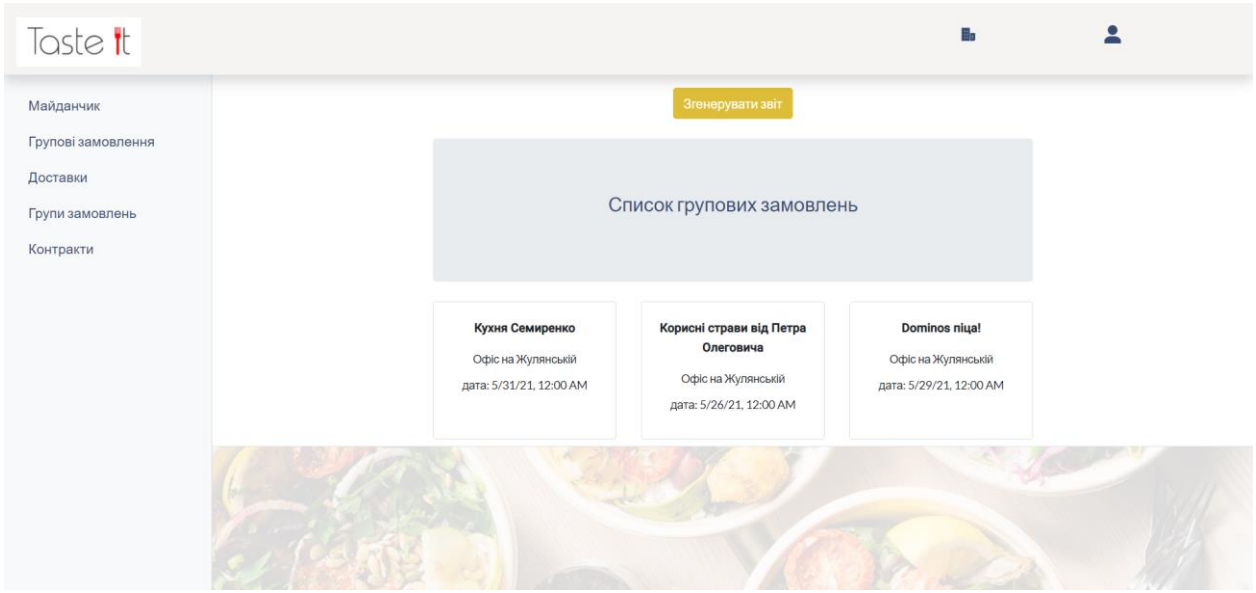

Рис. 20. Сторінка групових замовлень

На рис. 21 наведено сторінку груп замовлень обідів, якими керує конкретний менеджер.

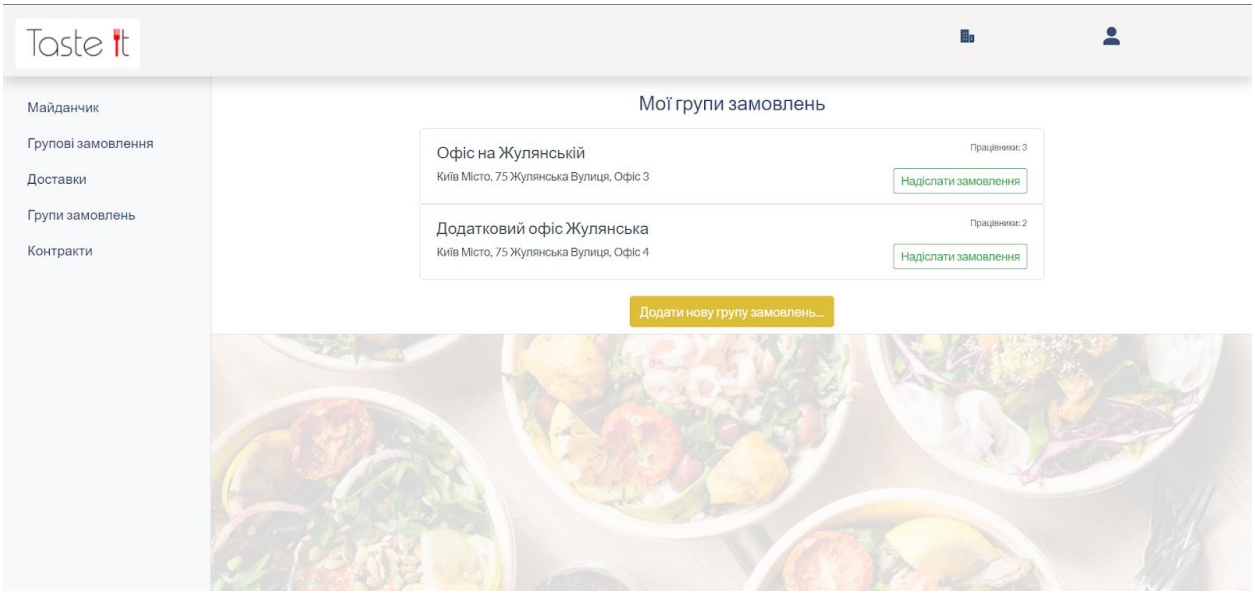

Рис. 21. Сторінка груп замовлень

Більш докладно ознайомитися з інтерфейсами менеджера можна в додатку В.

4.4. Інтерфейси працівника

На рис. 22 наведено сторінку доступних працівникові меню. На даній сторінці відображаються лише ті меню, що доступні компанії працівника, тобто такі меню, на які укладено активний контракт між компанією працівника та постачальниками. Неактивні меню не відображаються.

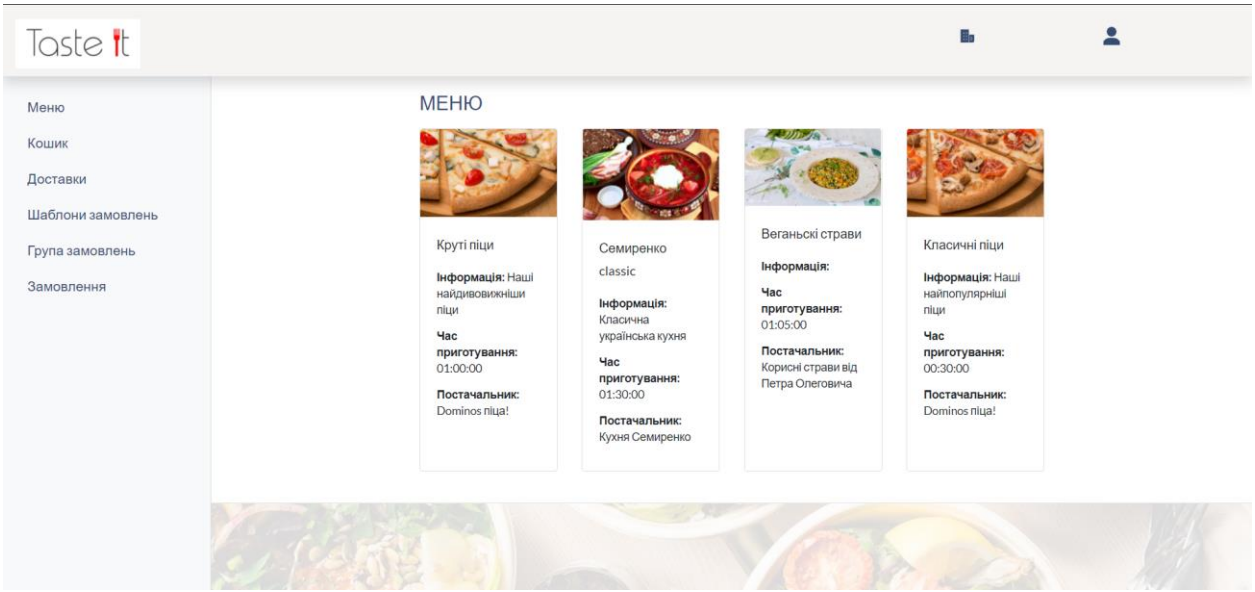

Рис. 22. Сторінка доступних меню для працівника

На рис. 23 наведено сторінку управління кошиком працівника. За допомогою даного інтерфейсу працівник може як зробити замовлення, так і зберегти його як шаблон для подальшого використання.

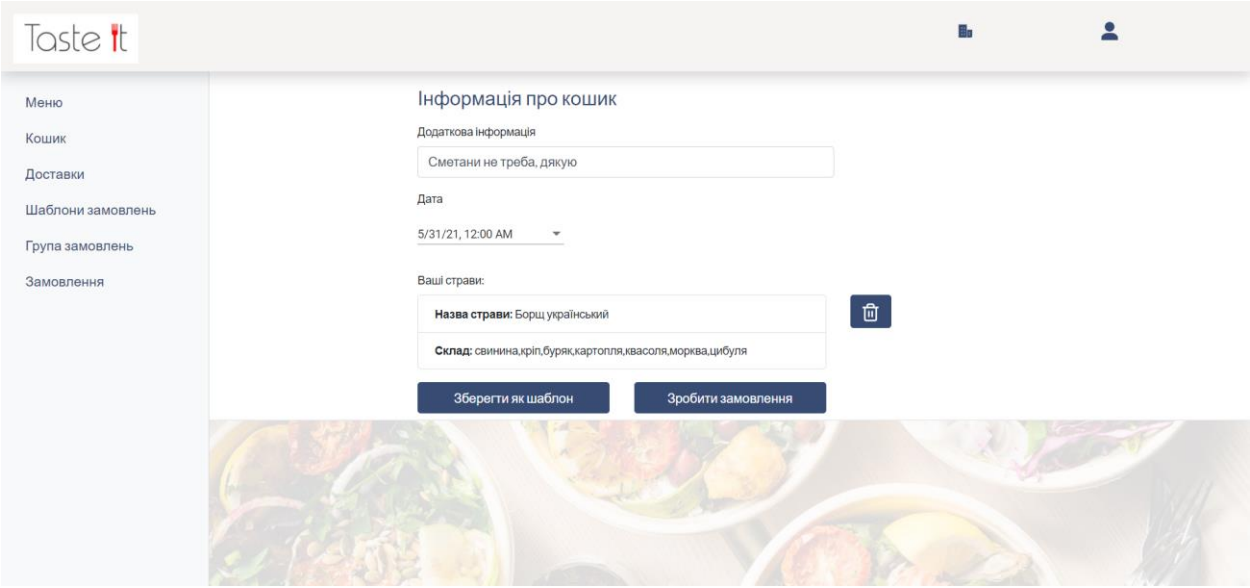

Рис. 23. Сторінка кошика працівника

Більш докладно ознайомитися з інтерфейсами працівника можна в додатку Г.

ВИСНОВКИ

В роботі було розглянуто проблему автоматизації процесу корпоративної доставки їжі, виконано загальний огляд onion-архітектури для серверної частини системи та обґрунтувано доцільність вибору такої архітектури, розглянуто односторінкові архітектуру для клієнтської частини додатку та обґрунтувано доцільність вибору такої архітектури.

Метою роботи було створення програмного засобу для автоматизації процесу замовлення їжі для корпоративного використання.

В результаті було розроблено програмний засіб для автоматизації процесу замовлення їжі для корпоративного використання, а саме виконано наступні завдання:

- визначено ключові особливості, що демонструють користь додатку з-поміж інших подібних засобів;
- проаналізовано можливі ролі користувачів програмного засобу, які забезпечують зручність використання системи кожним окремим користувачем та дозволяють чітко розділити дії користувачів системи;
- визначено особливості кожної ролі в системі;
- сформульовано вимоги до системи: функціональні з боку кожної ролі, нефункціональні;
- спроектовано та розроблено систему, що відповідає сформульованим вимогам;
- окреслено шляхи подальшого розвитку системи у вигляді додаткових вимог.

ДЖЕРЕЛА ПОСИЛАННЯ

1. SPA vs MPA: The definitive guide for decision makers [Електронний ресурс] – Режим доступу до ресурсу: https://www.mindk.com/blog/single-page-applications-the-definitive-guide/

2. Angular Docs [Електронний ресурс] – Режим доступу до ресурсу: https://angular.io/docs

3. OWASP – Cross Site Scripting (XSS) [Електронний ресурс] – Режим доступу до ресурсу: https://owasp.org/www-community/attacks/xss/

4. Mikowski M. Single Page Web Applications / Michael S. Mikowski, Josh Powell – 2013 – 432 с.

5. Robert C. Martin Clean Architecture: A Craftsman's Guide to Software Structure and Design / Robert C. Martin : Prentice Hall, 2017 – 432 с. – (Robert C. Martin Series)

6. Robert C. Martin Clean Architecture: A Code of Conduct for Professional Programmers / Robert C. Martin – 240 с. – (Robert C. Martin Series)

7. Daniel Marbach Chop Onions Instead of Layers in Software Architecture [Електронний ресурс] / Daniel Marbach – Режим доступу до ресурсу: Режим доступу до ресурсу: https://www.methodsandtools.com/archive/onionsoftwarearchitecture.php

8. Стаття The Clean Architecture [Електронний ресурс] – Режим доступу до ресурсу: Режим доступу до ресурсу: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

9. Моделі життєвого циклу додатків [Електронний ресурс] – Режим доступу до ресурсу: https://evergreens.com.ua/ru/articles/software-development-metodologies.html

10. Pandian, C. Ravindranath Applied Software Risk Management: A Guide for Software Project Managers / Pandian, C. Ravindranath – Auerbach Publications, 1st edition, 2006 – 264 с.

ДОДАТОК А. ЗАГАЛЬНІ ІНТЕРФЕЙСИ

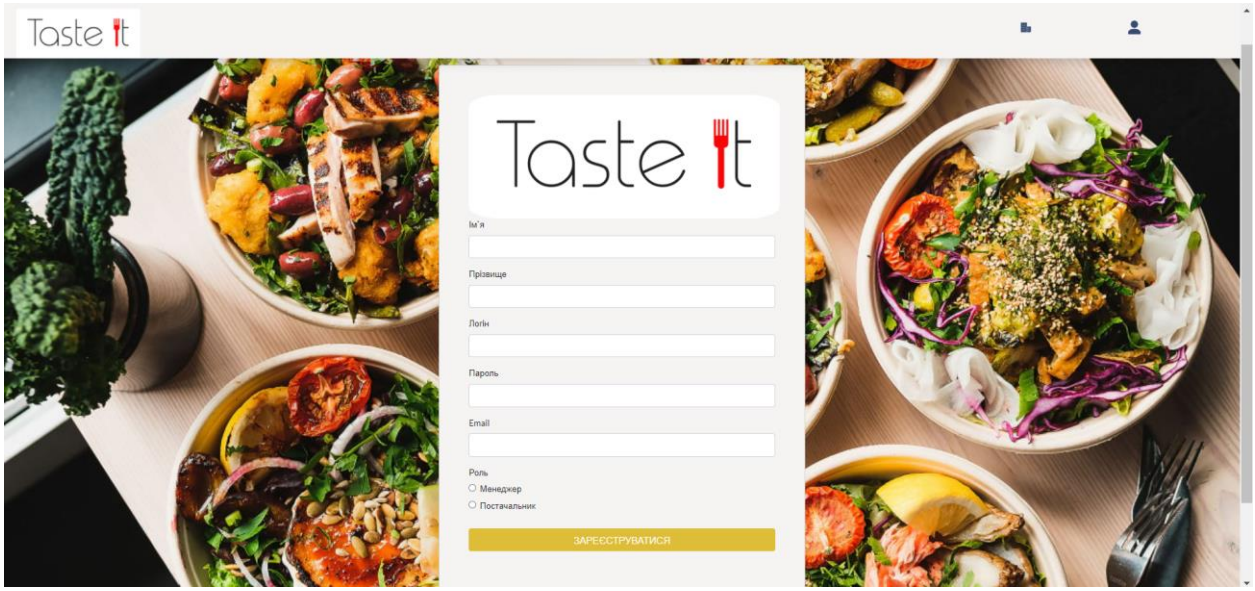

Рис. 24. Сторінка реєстрації

ДОДАТОК Б. ІНТЕРФЕЙСИ ПОСТАЧАЛЬНИКА

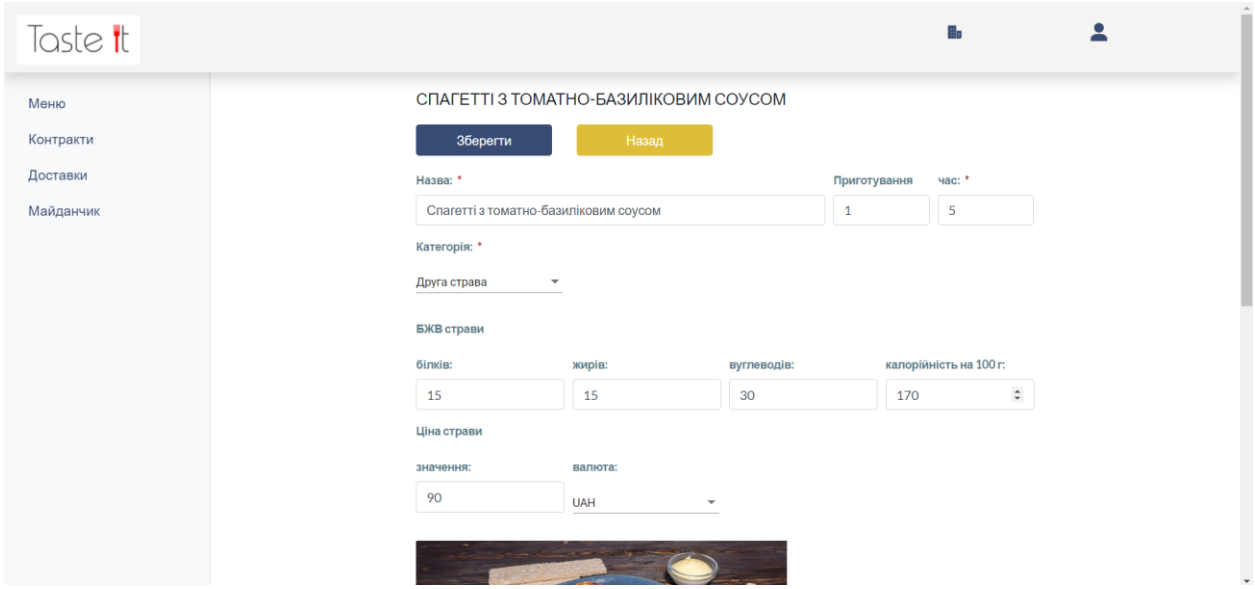

Рис. 25. Сторінка детальної інформації про страву

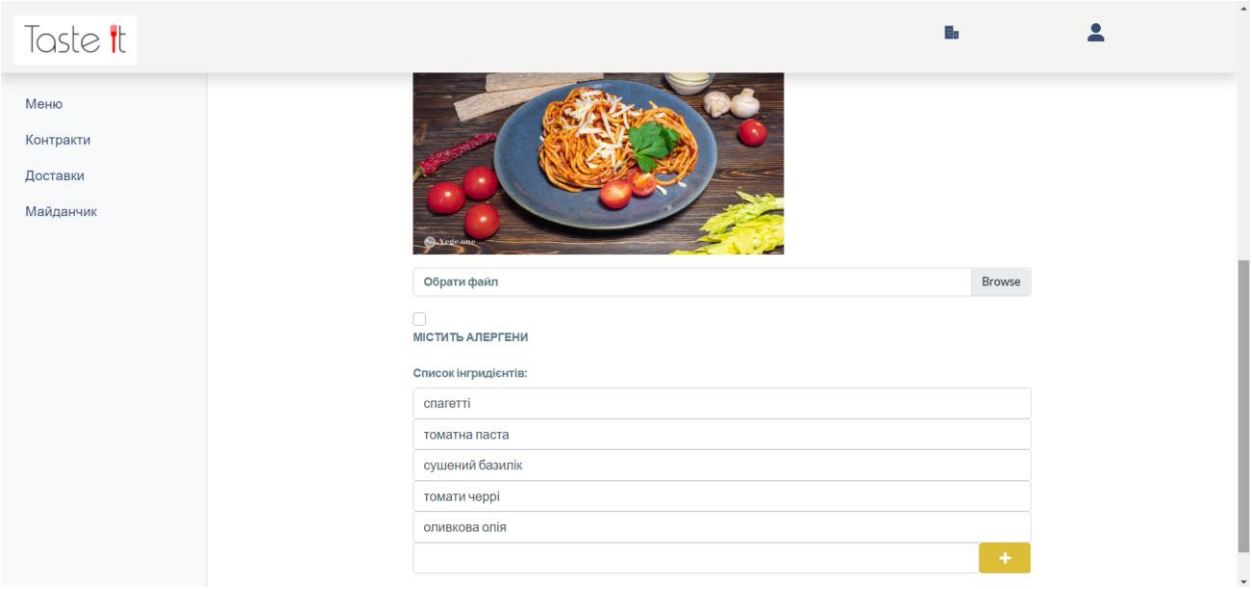

Рис. 26. Сторінка детальної інформації про страву (продовження)

ДОДАТОК В. ІНТЕРФЕЙСИ МЕНЕДЖЕРА

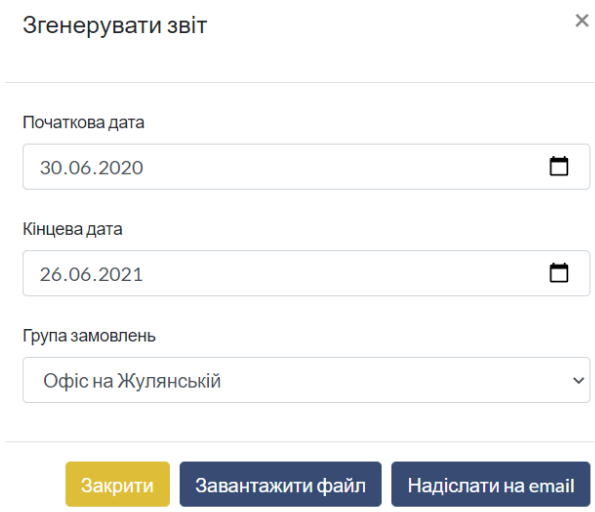

Рис. 27. Форма для заповнення фільтрів звіту менеджера

A	B	C	D	E
Період: 30.06.2020 - 26.06.2021				
Компанія: Microsoft Україна				
Менеджер: Максим Шевченко				
Група замовлень: Офіс на Жулянській				
Дата замовлення	**Працівник**	**Постачальник**	**Меню**	**Сума, грн**
26.05.2021	Катерина Брамс	Корисні страви від Петра Олеговича	Веганьскі страви	140
29.05.2021	Дмитро Опанасенко	Dominos піца!	Круті піци	627
29.05.2021	Анастасія Бучельницька	Dominos піца!	Круті піци	229

Рис. 28. Звіт

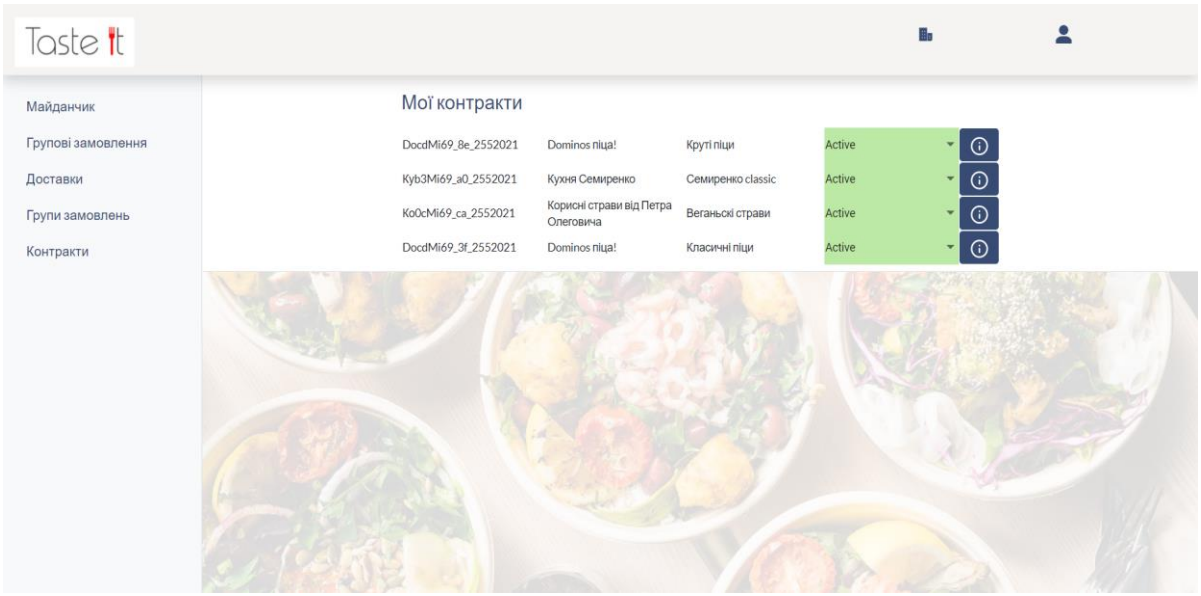

Рис. 29. Сторінка контрактів

ДОДАТОК Г. ІНТЕРФЕЙСИ ПРАЦІВНИКА

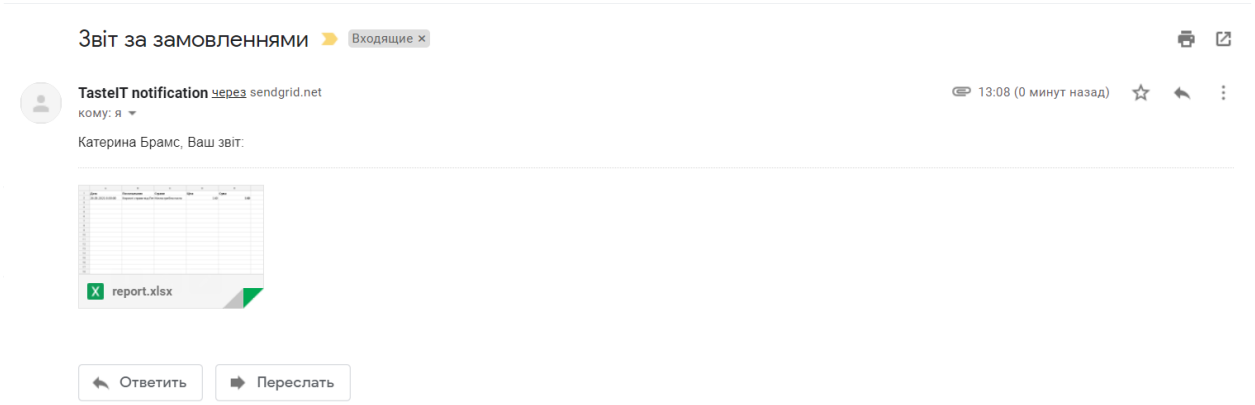

Рис. 30. Надісланий на пошту лист із звітом

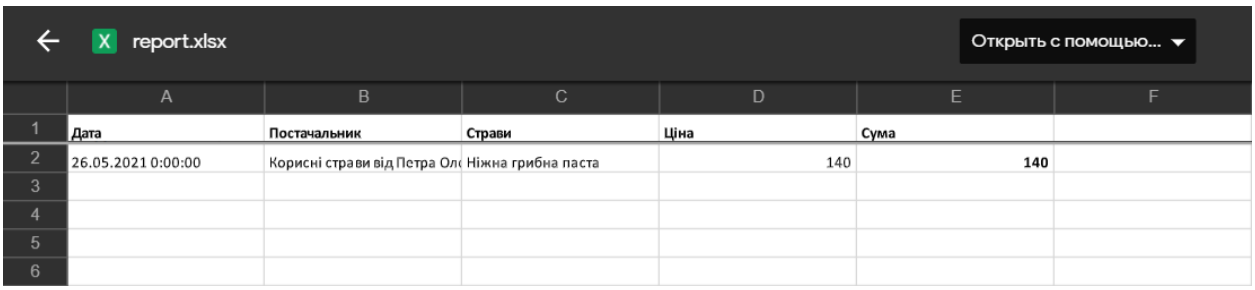

Рис. 31. Звіт