



Приватний заклад вищої освіти
Одеський технологічний університет «ШАГ»
Кафедра інформаційних технологій та фундаментальної підготовки

Випускна кваліфікаційна робота бакалавра

**Інформаційна система управління підрозділами розподілених
комерційних проектів
Mozaik (на прикладі ChoiceQR)**

на здобуття бакалаврського рівня вищої освіти
зі спеціальності «122 Комп'ютерні науки»

Виконавці проєкту:

№	П.І.Б.	Група
1	Дубська Олена Владиславівна	КН-Д-211
2	Белова Альона Ігорівна	КН-Д-211
3	Белов Максим Ігорович	КН-П-213
4	Ілечко Сергій Сергійович	КН-П-213
5	Лічкун Микита Вікторович	КН-П-213

Автор звіту

_____ Белов Максим Ігорович

РЕЗУЛЬТАТИ ЕКСПЕРТИЗИ

Експерт	П.І.Б.
Науковий керівник	доц. Самойленко Д. М.

АНОТАЦІЯ

УДК 004.9

Б-43

Белов М. І. Інформаційна система управління контентом закладів харчування: Автореферат випускної кваліфікаційної роботи на здобуття бакалаврського рівня вищої освіти зі спеціальності «122 Комп'ютерні науки». - Одеса: ОТУШ, 2025 - 20 с.

Бакалаврська робота присвячена дослідженню та розробці інформаційної системи, головне призначення якої полягає в тому, щоб дозволити закладам харчування створювати, редагувати та актуалізувати цифрові меню в зручному форматі. Основним завданням системи є забезпечення гнучкого інструменту для створення, структурування та оновлення електронних меню, доступних для редагування адміністрацією закладу та перегляду відвідувачами через QR-коди та посилання.

Актуальність теми обумовлена зростаючою потребою у цифровізації обслуговування клієнтів у сфері харчування, а також необхідністю оперативного оновлення меню без залучення сторонніх спеціалістів. Запропоноване рішення дозволяє автоматизувати процес керування асортиментом страв, спрощує роботу персоналу та покращує зручність взаємодії кінцевого користувача з меню.

Платформа складається з наступних частин, кожна з яких представляє певну функціональність:

- Авторизація
- Автентифікація
- Реєстрація
- Створення ієрархії онлайн меню
- Створення страв
- Зберігання загальних налаштувань онлайн меню
- Налаштування та генерація QR коду з посиланням на онлайн меню
- Редагування та зберігання налаштувань зовнішнього вигляду онлайн меню
- Керування зображеннями страв та іконок
- Додавання коментарів до медіафайлів
- Фільтрація страв за певними критеріями

ЗМІСТ

ВСТУП _____	4
Технічне завдання до проекту _____	5
РОЗДІЛ 1. Загальна архітектура програмної частини _____	9
РОЗДІЛ 2. Реалізація серверної частини _____	13
РОЗДІЛ 3. Інструменти для розробників _____	16
ВИСНОВКИ _____	18
ПЕРЕЛІК ПОСИЛАНЬ _____	20

ВСТУП

Зі зростанням та поширенням інформаційних технологій дедалі більше послуг переходить у цифрове середовище. Комунікаційні технології виступають незамінним засобом соціально-економічного розвитку та одним із ключових чинників інноваційного прогресу економіки.

Сфера обслуговувань також постійно еволюціонує, зокрема в частині цифрових рішень для закладів харчування. Якщо раніше меню існувало лише у вигляді паперових носіїв, то сьогодні усе частіше використовуються цифрові аналоги. Вони дають змогу власникам швидко змінювати інформацію, додавати нові позиції та стилістично відокремитись від решти інших закладів, що дає змогу автоматизувати процес взаємодії з клієнтами. Особливої популярності набули QR-меню, як зручне рішення для гостей, так і для працівників закладу.

Детальне технічне завдання до проєкту наводиться далі. У рамках даного звіту розглядається серверна частина, що являє собою API додаток, розміщений на хмарному середовищі Azure, та загальна архітектурна проєкту. Звіт складається з трьох розділів у яких описується загальна архітектура системи, деталі створення серверної частини та інструменти, які надані розробникам для роботи з проєктом. Висновки відбивають результати, що досягнуті у рамках даної частини проєкту, а також інших учасників проєкту.

ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЄКТУ

Загальний опис.

Програмне забезпечення (ПЗ) призначене для створення незалежних інформаційних системи, яка забезпечує цифрове управління меню в закладах харчування.

Головна особливість системи полягає у поєднанні трьох незалежних частин:

- Мобільного цифрового меню (для гостей закладу);
- Адміністративної панелі (для власника або адміністратора закладу, що дозволяє керувати усім вмістом меню);
- Лендінга – сторінки, яка закликає власника харчового закладу спробувати продукт.

Призначення:

ПЗ орієнтується на забезпечення зручного механізму взаємодії між кінцевим споживачем (відвідувачем закладу) та адміністративною частиною (представником закладу) шляхом візуалізації та керування інформаційними одиницями – категоріями, позиціями меню, модифікаторами, описами, інгредієнтами тощо. Сам процес створення та керуванням ресторану повинен бути швидким та інтуїтивним

Вимоги до функціональності:

Реєстрація користувачів:

Функціонал реєстрації користувача покликаний забезпечити зручний та інтуїтивно зрозумілий спосіб створення облікового запису для взаємодії з CMS-системою. На цьому етапі користувач має ввести такі дані, як:

- Назва ресторану – що дозволяє ідентифікувати об'єкт обслуговування та використовується для створення піддомену сайту ресторану;
- Логін – унікальний ідентифікатор користувача в рамках ресторану;
- Телефон – як засіб оперативного контакту;
- Електронна пошта – для можливості підтвердження дій та відновлення доступу;
- Пароль – обов'язково складний, для забезпечення належного рівня безпеки.

Вхід в систему:

Для входу у систему користувач вводить свої дані у відповідну форму. Необхідно зазначити:

- Піддомен ресторану;

- Логін або електронну пошту;
- Пароль.

Лендінг (Початкова сторінка)

Головна сторінка системи Mosaic орієнтована переважно на бізнес-користувачів – власників закладів харчування, і має за мету ознайомлення з сервісом, підвищення зацікавленості та швидку навігацію. Основна інформація представлена в мультирівневому форматі з якорями та кнопками в хедері. Передбачена можливість, ознайомлення з ключовими перевагами системи.

Адмін панель власника ресторану (CMS):

Основне робоче місце для налаштування ресторану та керування його наповненням. Повинна забезпечити усі необхідні засоби для

- Керування ієрархією меню, а саме:
 - 🌐 Розділами в межах ресторану, що дозволяє:
 - Змінювати назву розділу;
 - Додавати або редагувати його опис;
 - Керувати його видимістю (публічна/прихована);
 - 🌐 Категоріями в межах розділу, що передбачає:
 - Зміну назви категорії;
 - Керування її видимістю для клієнтів;
- Керування страв в межах категорії з можливостями вказати/додати:
 - 🌐 Фото (до 4 штук, вага до 2 МБ, можливість вибору обкладинки);
 - 🌐 Назву, опис, калорійність, БЖВ;
 - 🌐 Вагу або об'єм в альтернативних одиницях (грам, мілілітр тощо);
 - 🌐 Ціну та знижку;
 - 🌐 Видимість страви у клієнтському інтерфейсі.
- Налаштування зовнішнього вигляду онлайн меню
- Керування загальною інформацією про ресторан, зокрема:
 - 🌐 Назва, опис, адреса закладу;
 - 🌐 Основна валюта та мова обслуговування;
 - 🌐 Контактна пошта та телефони;
 - 🌐 Favicon сайту;

- Посилання на соціальні мережі;
- Графік роботи по кожному дню тижня.
- Створення та кастомізація зовнішнього вигляду QR-коду:
 - Стиль маркерів, колір, фон, прозорість;
 - Текст над або під кодом;
 - Рамка, логотип, розмір.
 - Зберігання налаштувань

Перегляд меню

Місце, з яким взаємодіють усі відвідувачі харчового закладу через посилання або QR-код. Має реалізовувати можливість:

- Фільтрацію за ціною, типом приготування, часом, вагою;
- Пошук страв;
- Вибір розділу та категорії;
- Перегляд детальної інформації про страви, яка включає:
 - Назву, фото, вагу, опис, склад, час приготування;
 - Рекомендації щодо схожих страв.
- Отримання необхідної інформації про заклад. В цей перелік входять:
 - Контактні дані (телефон та соцмережі),
 - Години роботи.
 - Адресу закладу.

Вимоги до функціональності. Опціональна частина

Мітки для відображення особливостей страв. Користувач має змогу створювати власні бейджі відповідно до концепції закладу.

Можливість вказання часу роботи розділу — наприклад, для сніданків або ланчів.

Встановлення іконок категорії.

Підтримка різних мов інтерфейсу.

Відновлення паролю

Додаткова інформація про страви. Поглиблені опції:

- Харчові уподобання,
- Наявність алергенів,
- Спосіб приготування;
- Присвоєння бейджів для фільтрації (наприклад, "Веганське" або "Гостре").

Деталізація оформлення елементів та стилів. Поглиблені можливості редагування:

- Елементів меню, карток, категорій;
- Розширене редагування кнопок;
- Налаштування фону (включно з .svg паттернами);
- Вибір готових тем, їх кастомізація та скидання;
- Стани (hover, default, pressed).

Перемикання теми CMS (світла/темна).

Функціональність для перегляду обраних страв (Кошик), їх редагування та видалення. Показує:

- Рекомендовані страви;
- Загальну вартість;
- Кнопку "Замовити".

Оформлення замовлення, де користувач обирає:

- Тип замовлення (доставка, бронювання, самовивіз);
- Персональні дані (ПІБ, телефон), адреса, час кількість персон, побажання;
- Окремі форми збору інформації для різних способів отримання замовлення;
- Спосіб оплати (готівка або картка);
- Суму замовлення;.

Архітектура

ПЗ має бути спроектоване за клієнт-серверною архітектурою з розподіленими вузлами:

- Mockup – візуальна модель інтерфейсу користувача із зазначенням зв'язків і переходів між різними частинами інтерфейсу.
- Backend – серверне програмне забезпечення, що забезпечує збереження та оброблення поточних даних, взаємодію з базою даних. Надає програмний інтерфейс взаємодії (API) з іншими вузлами
- Frontend – додаток, доступний з веб-простору (сайт), що забезпечує візуальний інтерфейс користувачів

Усі модулі ПЗ повинні мати засоби

- тестування функціональності,
- розгортання на новому пристрої (інсталяції)

Технології

Зберігання вихідних кодів ПЗ повинно здійснюватись за допомогою системи контролю версій (вибір конкретної системи узгоджується групою проекту). Структура файлів та каталогів обирається на розсуд команди, але повинна максимально відбивати архітектуру ПЗ

Для кожного з модулів рекомендується (з можливістю часткової зміни) наступні технології створення:

- Backend – .NET або Java, або NodeJS технології з хмарним розміщенням (Azure, AWS, Google Cloud),
- Frontend – серверні технології (ASP, Razor Pages, JSP) або React чи Angular
- Mockup - визначається ментором з дизайну

РОЗДІЛ 1. ЗАГАЛЬНА АРХІТЕКТУРА ПРОГРАМНОЇ ЧАСТИНИ

Розділення системи на компоненти:

У відповідності до технічного завдання (ТЗ) система була спроектована за клієнт-серверною розподіленою архітектурою. У складі системи було умовно відокремлено наступні архітектурні частини:

- Frontend (ресторан).
- Backend (ресторан)
- Frontend (лендінг та CMS).
- Backend (CMS)

Система було розбита на дві пари (Frontend та Backend), які відносяться до частини платформи управління контентом (CMS) та онлайн меню.

Причин для розділення Frontend частини три:

1. Підтримка стилів для двох частин. Лендінг та CMS мають чіткий брендовий стиль, у той час як сайт з онлайн меню повинен бути максимально гнучким у плані кастомізації.
2. Потенційна відмінність бібліотек дозволить краще підтримувати мінімальний розмір файлів, які передаються користувачу
3. Можливості оптимізації сайтів ресторану, як, наприклад, компіляція файлів розмітки та стилів для зменшення даних, що передаються між Frontend та Backend частиною

Причин для розділення Backend частини наступні:

1. Потенційна можливість керувати декількома ресторанами одним користувачем

2. Можливість реплікації ресторанної частини без дублювання чутливих даних

Також є одна загальна причина для обох частин:

1. Розмежування аутентифікації та авторизації адмін панелі та сайту ресторану. Сутності “адміністратор ресторану” та “відвідувач ресторану” не матимуть нічого спільного, оскільки мають доступ до різних частин системи. Таке розділення дозволить нам потенційно мати окрему структуру для зареєстрованих користувачів закладів, що зменшує вектор атаки та спрощує підтримку коду.

Система контролю версій:

В якості системи контролю версій був використаний GIT як індустріальний стандарт. Код розташований на двох приватних GitHub репозиторіях – RestaurantSaasBackend та RestaurantSaasFrontend. Був обраний шлях двох монорепо, що дозволило розмежувати зону відповідальності в межах команди без шкоди для зручності розробки. Структура монорепо також дозволяє дотримання принципу *DRY (Don't Repeat Yourself)*, використовуючи загальні модулі для обох частин

В рамках стратегії розгалуження була обрана стратегія GitHub Flow[1] як простий та ефективний спосіб організації командної розробки. Були також затверджені вимоги щодо оформлення Pull Requests та процедури перегляду коду, щоб підтримувати дисципліну розробки, що дуже важливо для підтримки якості коду.

Стратегія розгортання:

За середовище розгортання був обраний Azure через гнучкість та гарну інтеграцію з інструментами, які були використані для розробки.

Кожна з чотирьох частин проекту представляє собою моноліт, який розгортається як єдине ціле. Причини такого підходу дві:

- Невеликий розмір команди, що не дозволяє ефективно використовувати альтернативи по типу мікросервісної архітектури
- Відносно невеликий потенціал завантаження кожного окремого сайту ресторану. Очікуваний трафік для харчових закладів варіюється від декілька сотень до декілька тисяч на день, що значно менше за платформи інших форматів. Сайт rozetka.com.ua, до наприкладу, має приблизно 1.4 млн відвідувачів щодня[2]

Для вирішення проблем швидкодії передбачена можливість відокремлювати завеликі сайти ресторанів на окремі вузли

На момент написання звіту весь проєкт знаходиться на одній Azure Virtual Machine у вигляді наступних контейнерів:

```
mozaik_user@main-node: ~
mozaik_user@main-node:~$ docker ps --format "table {{.ID}}\t{{.Names}}\t{{.Image}}"
CONTAINER ID      NAMES                                IMAGE
0a168c12ba50      frontend-reverse-proxy              belomaximka/restaurant-reverse-proxy:latest
8dbf722aace9      customer-backend                    belomaximka/restaurant-backend:latest
6906e6a9eedd      main-backend                        belomaximka/restaurant-main-backend:latest
f95945d96bcf      postgres-db                         postgres
15937579dd8a      watchtower                          containrrr/watchtower
mozaik_user@main-node:~$
```

Рисунок 1.1 – Лог виконання команди для перегляду активних контейнерів

- frontend-reverse-proxy – зворотний проксі, який або роздає статичні файли фронтенд частини обох сайтів, або перенаправляє трафік на API в залежності від URL.
- customer-backend – API, який використовується сайтом онлайн меню
- main-backend – API, який використовується при роботі з CMS
- postgres-db – база даних, до яких під'єднані обидві API
- watchtower – контейнер, який автоматично оновлює інші контейнери, якщо бачить новий докер образ.

Для контейнерів налаштовані томи та зміни середовища, тому після перезавантаження віртуальної машини сервіси повертаються в робочий стан.

Для дотримання норм безпеки порти для віртуальної машини обмежені лише до тих, які необхідні: HTTPS/HTTP для всіх, SSH та PostgreSQL обмежені по IP лише для розробників.

Priority ↑↓	Name ↑↓	Port ↑↓	P... ↑↓	Source ↑↓	Desti... ↑↓	Action ↑↓
Inbound Security Rules						
300	SSH	22	TCP		Any	✓ Allow
320	HTTPS	443	TCP	Any	Any	✓ Allow
340	HTTP	80	TCP	Any	Any	✓ Allow
350	AllowAnyPostgreS...	5432	TCP		Any	✓ Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNet...	✓ Allow
65001	AllowAzureLoadBal...	Any	Any	AzureLoadBalancer	Any	✓ Allow
65500	DenyAllInBound	Any	Any	Any	Any	✗ Deny

Рисунок 1.2 – Налаштування правил для вхідного трафіку віртуальної машини

Домен та TLS сертифікати:

Ми маємо придбаний на namecheap.com домен *mozaik.rest*, який прив'язали до серверу. Оскільки новостворені сайти ресторанів мають адресу у вигляді піддомену (наприклад, my-restaurant.mozaik.rest), для підтримки безпечного, зашифрованого TLS з'єднання ми використовували безкоштовний wildcard сертифікат, який надає сервіс Let's Encrypt. Нажаль, Namecheap не дозволяє автоматичного поновлення wildcard сертифікатів (оскільки вони вимагають dns верифікацію), тому для цього необхідно поновлювати сертифікат вручну за допомогою certbot, витрачаючи приблизно 10 хвилин кожні 90 днів.

CI/CD:

На кожен репозиторій налаштований GitHub Actions, який на кожен push у гілку master будує образ та завантажує його у безкоштовний Docker реєстр. Watchtower, по який ми згадували раніше, відповідальний за оновлення контейнерів до останньої версії образів.

Це дозволило нам мати всі останні зміни на сервері, що дає змогу переконатися, що вся функціональність працює і готова для використання, та отримувати ранній зворотній зв'язок від учасників команди або знайомих.

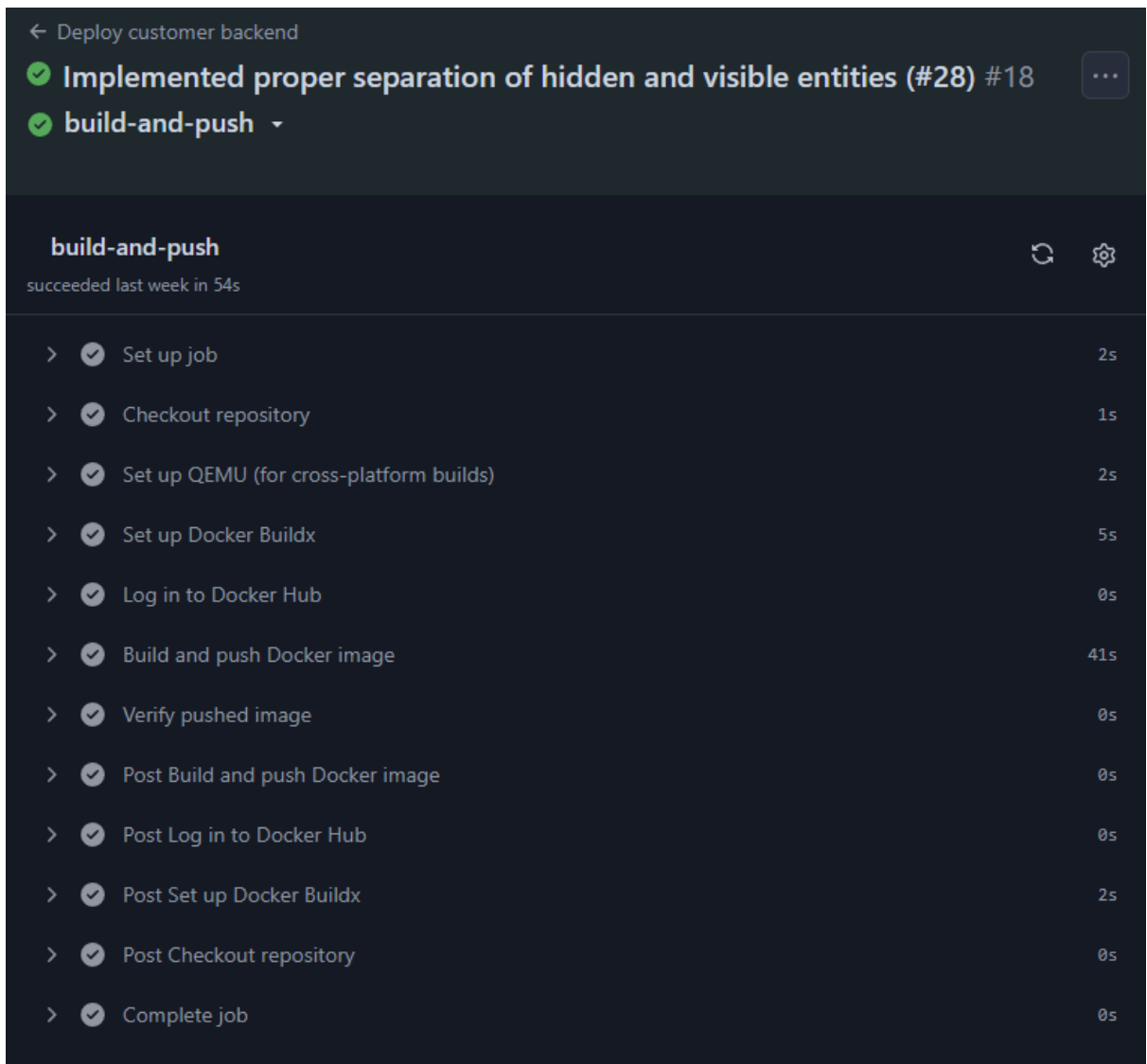


Рисунок 1.2 – Робота CI/CD пайплайнів

РОЗДІЛ 2. РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ

Архітектура та технології:

Серверна частина реалізована за архітектурою REST API з використанням наступних технологій:

- .NET 9 як платформи для побудови API
- [ASP.NET](#) Core як основного фреймворку
- PostgreSQL в якості бази даних (БД).
- Entity Framework Core для взаємодії з БД.
- [ASP.NET](#) Core Identity для роботи з аутентифікацією та авторизацією
- Microsoft Visual Studio Community 2022 як середовище розробки
- ImageSharp для роботи з зображеннями
- Azure.Storage.Blobs. C# бібліотека для роботи з Azure Blob Storage
- DBeaver. Інструмент для прямого доступу до бази даних

Внутрішня архітектура серверної частини побудована за N-layer структурою, що логічно розділяє його на декілька частин:

- Application Layer – шар, в якому знаходяться контролери, middlewares, та загальні налаштування додатку.
- Business Logic Layer – шар, поділений на сервіси, в якому відбувається вся бізнес логіка
- Data Access Layer – шар, у якому відбувається налаштування сутностей EF Core та взаємодія з базою даних
- Shared – шар, спільний для CustomerAPI та MainAPI, в якому знаходяться спільні контракти та логіка

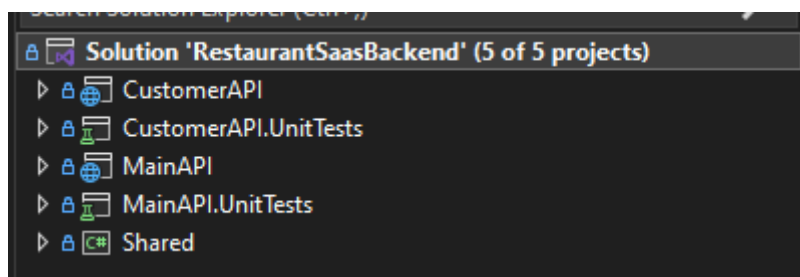


Рисунок 2.1 – Структура рішення серверної частини

Ця архітектура зарекомендувала себе як надійний варіант для середньо-маленьких проєктів, дозволяючи розподілити відповідальність між частинами проєкту.

При проектуванні класів були використані принципи SOLID, щоб забезпечити мінімальне зв'язування між класами. Dependency Injection та DI контейнери, які надаються фреймворком [ASP.NET](#) Core, дозволили тримати залежності між компонентами зрозумілими та контрольованими

База даних:

При проектуванні бази даних PostgreSQL були втілені перші три нормальні форми. Доступ до неї відбувався за допомогою Entity Framework Core, з якої ми активно використовували ORM функціональність та міграції, що дозволило керувати сутностями бази даних за допомогою підходу “Code First”. Для оптимізації роботи з EF Core були використані рекомендовані[3] підходи, такі як AsNoTracking, AsSplitQuery, Projection та інші

```
public Task<List<T>> GetAllVisibleMappedAsync<T>(int restaurantId, Expression<Func<Section, T>> mapper)
{
    return context.Sections
        .Where(s => s.RestaurantId == restaurantId
            && s.IsVisible
            && s.Categories.Any(c => c.IsVisible && c.Products.Any()))
        .OrderBy(s => s.Position)
        .Select(mapper)
        .ToListAsync();
}
```

```
public static readonly Expression<Func<Database.Entities.Categories.Category, CategoryDto>> AsDto =
    category => new CategoryDto
    {
        Id = category.Id,
        Name = category.Name,
        IsVisible = category.IsVisible,
        Position = category.Position,
        SectionId = category.SectionId,
        Products = category.Products
            .AsQueryable()
            .Select(ProductMappingExpressions.AsDto)
            .ToList()
    };
};
```

```
public static readonly Expression<Func<Database.Entities.Sections.Section, SectionFullDto>> AsFullDto =
    section => new SectionFullDto
    {
        Id = section.Id,
        Name = section.Name,
        Position = section.Position,
        Categories = section.Categories
            .AsQueryable()
            .OrderBy(c => c.Position)
            .Select(CategoryMappingExpressions.AsDto)
            .ToList()
    };
};
```

Рисунок 2.2 – Проекції DTO для оптимального отримання інформації з бази даних за допомогою Entity Framework Core

Автентифікація та авторизація:

Для роботи з адмін панеллю система вимагає від користувача бути автентифікованим, щоб уникнути несанкційних змін. MainAPI на кожен запит очікує куки *identity*, в якому знаходиться уся необхідна інформація. Значення куки зашифроване та підписане за допомогою Data Protection API (DPAPI)[6]. Отримати куки користувач може за адресою /api/accounts/sign-in, передавши значення домену ресторану, логін та пароль. Паролі зберігаються у базі даних у вигляді хешу, використовуючи PBKDF2 з HMAC-SHA512, 128-бітну сіль, 256-бітний підключ та 100000 ітерацій [7].

Куки налаштовані таким чином, щоб забезпечити достатній рівень безпеки та уникнути ряд атак, такі як CSRF.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite
identity	CfDJ...	mozaik.rest	/	2025-07-03T02:2...	718	✓	✓	Strict

Рисунок 2.3 – Демонстрація налаштувань Cookie

CustomerAPI має набір адрес для керування сутностями, всі вони захищені власне написаним фільтром, який вимагає мати у заголовку запита спеціальний ApiKey, який надсилає MainAPI після верифікації прав користувача.

```
private sealed class AuthorizeByApiKeyFilter(string apiKey) : IAuthorizationFilter
{
    0 references
    public void OnAuthorization(AuthorizationFilterContext context)
    {
        if (string.IsNullOrEmpty(apiKey) ||
            !context.HttpContext.Request.Headers.TryGetValue("X-API-Key", out var providedKey) ||
            !CryptographicOperations.FixedTimeEquals(
                Encoding.UTF8.GetBytes(providedKey!),
                Encoding.UTF8.GetBytes(apiKey)))
        {
            context.Result = new UnauthorizedResult();
        }
    }
}
```

Рисунок 2.4 – Фільтр авторизації на CustomerAPI за допомогою API ключа з використанням криптографічно стійких алгоритмів

Перевірка значень ключа використовує спеціальний метод перевірки, який захищає систему від атак за часом виконання.

Використання дуже популярного JWT механізму не надало би нам ніяких переваг, а тільки створило ризик виникнення безпекових вразливостей через підтримку власної логіки роботи з токенами. Саме тому ми обрали саме такий механізм автентифікації та авторизації, спираючись на вбудовані механізми браузеру та бібліотеки Identity.

Оптимізація зображень:

Для зберігання зображень використовується Azure Blob Storage, взаємодія з яким відбувається через офіційний пакет *Azure.Storage.Blobs*. Оскільки сервіс передбачає велику кількість зображень, було дуже важливо оптимізовувати їх перед збереженням. Для цього в нагоді була бібліотека *SixLabors.ImageSharp*, яка є безкоштовною для комерційного використання при умові грошового обігу до 1 мільйону доларів[4]. За допомогою неї зображення конвертуються у сучасний та ефективний формат WebP, який має широку підтримку[5]. При цьому зображення зменшуються до оптимального розміру, що подекуди призводить до оптимізації розмірів у ~50 разів.

РОЗДІЛ 3. Інструменти для розробників

Swagger:

Для того, щоб нові розробники мали змогу одразу зрозуміти, який набір операцій підтримує певний API, до двох проектів був доданий та налаштований Swagger



Рисунок 3.1 – Демонстрація захищених та публічних ендпоінтів через Swagger

Оскільки CustomerAPI використовує власну схему авторизації через API ключ, який Swagger не підтримує за замовчуванням, були додані спеціальні налаштування, які показують захищені ендпоінти, а також додають можливість вказати API ключ.

ApiKey (apiKey)

API Key needed to access protected endpoints. Example: `X-API-Key: your-key`

Name: X-API-Key

In: header

Value:

Authorize

Close

Рисунок 3.2 – Демонстрація можливості встановлення ApiKey для полегшення роботи з захищеними ендпоінтами через Swagger

Адаптація нових розробників:

Система побудована таким чином, щоб новий розробник міг максимально швидко підняти проект та почати локальну роботу з API. Обидві частини інформаційної системи мають інструменти, які полегшують роботу розробникам:

- Backend надає:
 - 🌐 Документацію стосовно того, як налаштувати зовнішні залежності для локальної роботи
 - 🌐 Можливість застосувати міграції для локальної бази даних
 - 🌐 Альтернативну реалізацію для роботи з файлами, яка використовує локальне сховище
 - 🌐 Попередньо налаштовані `appsettings.Development.json` для середовища розробки без жорстко закодованих секретів
- Frontend надає:
 - 🌐 Альтернативну конфігурацію для локальної розробки
 - 🌐 Список готових команд в `package.json`
 - 🌐 Спеціальну логіку, яка дозволяє використовувати параметри шляху замість піддоменів для розпізнавання закладів харчування

ВИСНОВКИ

Проведено огляд цифрових рішень для закладів харчування, виявлено перелік функціональностей, який має найбільшу цінність:

- Наповнення меню стравами з детальною інформацією;
- Персоналізація зовнішнього вигляду сайту з переліком страв;
- Персоналізація та генерація QR коду з посиланням на онлайн меню;
- Інструменти для навігації та пошуку на сторінці зі стравами;
- Актуалізація даних у реальному часі.

Задля досягнення результату, який був поставлений у технічному завданні, та чіткої організації робочих процесів для більш контрольованої системи роботи, була використана система Plane.so, яка дозволила розподілити ролі та відповідальність, а також контролювати строки виконання задач.

Були прийняті архітектурні рішення, які дозволили власникам харчових закладів створювати власні сайти керування контентом з мінімальною затримкою:

- Створення нових пустих ресторанів як запис в базі даних в рамках одного вузлу замість розгортання окремого;
- Закріплення сутностей за певним рестораном, ідентифікація певного закладу за піддоменом в адресі;
- Використання хмарних технологій для можливостей гнучкого підлаштування під різне навантаження.

Були враховані потенційні можливості для масштабування:

- Винесення завеликих харчових закладів на окремі вузли;
- Перенесення сайтів онлайн меню на технологію SSR з подальшим кешуванням сторінок.

Спроековано, реалізовано та випробувано інформаційну систему управління контентом закладів харчування. Система складена за розподіленою клієнт-серверною архітектурою та складається з трьох частин, розміщених за адресами:

- Mockup: <https://www.figma.com/file/d9ivM3L0t63EbYJjonjuk2>
- Backend: <https://github.com/BeloMaximka/RestaurantSaasBackend>
- Frontend: <https://github.com/BeloMaximka/RestaurantSaasFrontend>

Систему було розгорнуто на реальному домені та випробувано як членами команди, так і різними добровольцями. Результати випробування підтвердили загальну працездатність створеної системи.

Перспективи подальших досліджень полягають у:

- Впровадженні широкої системи кешування без шкоди для негайного зворотного зв'язку;
- Впровадженні засобів моніторингу для прийняття рішень оптимізації та масштабування;

- Розгляд переносу архітектури на gRPC для оптимізації трафіку та синхронізації контрактів;
- Реалізація опціональної функціональності, як відновлення паролю, підтвердження пошти, локалізація сутностей, підтримка додаткової інформації для страв тощо.

ПЕРЕЛІК ПОСИЛАНЬ

Використані джерела:

1. Офіційна документація описує простий гілкового робочого процесу GitHub Flow. URL: <https://docs.github.com/en/get-started/using-github/github-flow>
2. Аналітика трафіку та рейтинг інтернет-магазину Rozetka. URL: <https://www.similarweb.com/website/rozetka.com.ua/>
3. Рекомендації щодо ефективного формування запитів у Entity Framework Core. URL: <https://learn.microsoft.com/en-us/ef/core/performance/efficient-querying>
4. Інформація про комерційні ліцензії для бібліотек Six Labors,. URL: <https://sixlabors.com/pricing/>
5. Офіційний опис формату WebP від Google. URL: <https://developers.google.com/speed/webp>
6. Інформація про зберігання та роботу з cookie бібліотекою ASP.NET Core Identity. URL: <https://learn.microsoft.com/en-us/aspnet/core/security/data-protection/introduction>
7. Хешування паролів за допомогою ASP.NET Core Identity. URL: <https://github.com/dotnet/aspnetcore/blob/704f7cb1d2cea33afb00c2097731216f121c2c73/src/Identity/Extensions.Core/src/PasswordHasher.cs>

Використані ресурси:

1. ChatGPT. Використовувався для аналізу, розгляду різних підходів під час вирішення задач, написання скриптів. URL: <https://chatgpt.com>
2. ChoiceQR. Служив базою для аналізу інженерних рішень. <https://choiceqr.com>
3. Microsoft Learn. Документація до великої кількості інструментів від Microsoft. <https://learn.microsoft.com>
4. Документація PostgreSQL. <https://www.postgresql.org/docs>
5. Robert C. Martin and Micah Martin. Agile Principles, Patterns, and Practices in C#
6. Jeffrey Richter. CLR via C# (4th Edition) (Developer Reference)
7. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software