МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Острозька академія»
Економічний факультет
Кафедра економіко-математичного моделювання та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавра

на тему: «**Розробка системи управління замовленнями лаунж-бару «Replica»»**

Виконав: студент 4 курсу, групи КН-4
першого (бакалаврського) рівня вищої освіти
спеціальності 122 Комп'ютерні науки
освітньо-професійної програми «Комп'ютерні науки»
Тимощук Роман Васильович

Керівник: старший викладач кафедри ЕММІТ,
Клебан Юрій Вікторович

Рецензент: Front-end Developer "DOODLE" LLC,
Місай Володимир Віталійович

РОБОТА ДОПУЩЕНА ДО ЗАХИСТУ

Завідувач кафедри економіко-математичного моделювання та інформаційних технологій _____ (проф., д.е.н. Кривицька О.Р.)

Протокол № 11 від 18 травня 2023 р.

Острог, 2023

Міністерство освіти та науки України
Національний університет «Острозька академія»

Факультет: *економічний*

Кафедра: *економіко-математичного моделювання та інформаційних технологій*

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Ольга КРИВИЦЬКА
«____» _____20__ р.

З А В Д А Н Н Я
на кваліфікаційну роботу/проєкт студента

Тимощука Романа Васильовича

1. Тема роботи: Розробка системи управління замовленнями лаунж-бару «Replica»

керівник роботи Клебан Юрій Вікторович, старший викладач кафедри ЕММІТ

Затверджено наказом ректора НаУОА від "___"_____ 20__ року №___

2. Термін здачі студентом закінченої роботи: _____

3. Вихідні дані до роботи: для реалізації даного проєкту було використано наступний стек технологій: BlazorWASM, Entity Framework, JWT, ASP.NET, C#.

4. Перелік завдань, які належить виконати: реалізувати структуру проєкту з дотриманням чистої архітектури. Створити базу даних для подальшої обробки та зберігання інформації, яку потребує програмний застосунок. Реалізувати функціонал серверної частини проєкту, описавши всі контролери API та інтерфейси для обробки запитів. Додати авторизацію за допомогою JWT. Реалізувати клієнтську частину, створити сторінку авторизації та реєстрації користувачів в системі. Сторінки виводу продукції лаунж-бару, замовлення, підтвердження замовлення та особистого кабінету користувача.

5. Перелік графічного матеріалу: рисунки, таблиці

6. Консультанти розділів роботи:

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Клебан Ю. В., старший викладач	01.12.2023р.	01.12.2023р.
2	Клебан Ю. В., старший викладач	01.12.2023р.	01.12.2023р.
3	Клебан Ю. В., старший викладач	01.12.2023р.	01.12.2023р.

7. Дата видачі завдання: 01.12.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів	Примітка
1	Затвердження теми роботи/проєкту	31.12.2022 р.	
2	Аналіз сфери роботи лаунж-барів	31.12.2022 р.	
3	Огляд проблематики процесу формування замовлень	04.01.2023 р.	
4	Написання технічного завдання	07.01.2023 р.	
5	Ознайомлення з технологіями	15.01.2023 р.	
6	Вибір архітектурного рішення	17.01.2023 р.	
7	Створення проєкту, реалізація структури архітектурного рішення	19.01.2023 р.	
8	Проєктування бази даних	30.01.2023 р.	
9	Реалізація генерації JWT, створення API реєстрації та авторизації	12.02.2023 р.	
10	Написання API для всіх необхідних функій проєкту	24.02.2023 р.	
11	Тестування API	18.03.2023 р.	
12	Реалізації валідації даних	20.03.2023 р.	
13	Написання front-end	30.03.2023 р.	
14	Налагодження взаємодії сервера та клієнта	15.05.2023 р.	
15	Тестування	28.05.2023 р.	
16	Здача кваліфікаційної роботи/проєкту на кафедру	31.05.2023 р.	

Студент: _____ Роман ТИМОЩУК

Керівник кваліфікаційної роботи: _____ Юрій КЛЕБАН

АНОТАЦІЯ
кваліфікаційної роботи/проєкту
на здобуття освітнього ступеня бакалавра

Тема: *Розробка системи управління замовленнями лаунж-бару "Replica"*

Автор: *Тимощук Роман Васильович*

Науковий керівник: *Клебан Юрій Вікторович, старший викладач.*

Захищена «……»…… 20__ року.

Пояснювальна записка до кваліфікаційної роботи: *61 с., 23 рис., 0 табл., 23 джерел.*

Ключові слова: *фреймворк, лаунж-бар, JWT, Blazor WASM, система управління.*

Короткий зміст праці:

Завданням кваліфікаційної роботи/проєкту, було розробка системи управління замовленнями лаунж-бару "Replica". Зазначений проєкт являє собою реалізацію системи управління замовленнями для лаунж-бару, з можливістю авторизуватися в системі, а також переглядати список товарів та послуг які пропонує заклад. Основним завданням проєкту є надання можливості авторизації в системі та перегляду списку товарів та послуг, які пропонує заклад.

Система управління замовленнями лаунж-бару виконує роль зручного інструменту взаємодії між потенційними клієнтами закладу та його персоналом. Вона надає клієнтам зручний інтерфейс для створення та керування замовленнями послуг та продукції лаунж-бару. Завдяки системі управління замовленнями, клієнти зможуть легко здійснювати процес замовлення, вибираючи необхідні товари та послуги з переліку, який надає заклад. Вони також зможуть відстежувати статус своїх замовлень та отримувати повідомлення про їхню обробку.

Для персоналу лаунж-бару система управління замовленнями надає зручний інтерфейс для обробки та керування замовленнями. Вона дозволяє персоналу підтверджувати та опрацьовувати замовлення, а також забезпечує можливість керування меню, включаючи додавання, редагування та видалення продуктів.

Основною метою розробки системи управління замовленнями лаунж-бару "Replica" є покращення процесу обслуговування клієнтів та оптимізація роботи закладу шляхом автоматизації та ефективного керування замовленнями. Ця система допоможе підвищити

задоволеність клієнтів, забезпечити швидку обробку замовлень та полегшити роботу персоналу лаунж-бару.

The task of the qualification work/project was to develop an order management system for the lounge bar "Replica". This project is the implementation of an order management system for a lounge bar, with the ability to log in to the system, as well as view the list of goods and services offered by the institution. The main task of the project is to provide the ability to authorize in the system and view the list of goods and services offered by the institution.

The lounge bar order management system serves as a convenient tool for interaction between potential customers of the establishment and its staff. It provides customers with a convenient interface for creating and managing orders for services and products of the lounge bar. Thanks to the order management system, customers will be able to easily carry out the ordering process by choosing the necessary goods and services from the list provided by the establishment. They will also be able to track the status of their orders and receive notifications about their processing.

For the lounge bar staff, the order management system provides a convenient interface for processing and managing orders. It allows staff to confirm and process orders, and provides the ability to manage the menu, including adding, editing, and deleting products.

The main goal of developing the order management system for the lounge bar "Replica" is to improve the customer service process and optimize the work of the establishment by automating and efficiently managing orders. This system will help to increase customer satisfaction, ensure fast order processing and facilitate the work of the lounge bar staff.

ЗМІСТ

ВСТУП

У сучасному світі автоматизація є одним з найбільш актуальних трендів. Люди стають все більш зайнятими та прагнуть до спрощення процесів, тому для бізнесу дедалі більше потрібні ефективні системи управління, які допомагають зменшити навантаження на працівників та підвищити рівень задоволеності клієнтів. Одним зі сфер, де це особливо актуально, є готельно-ресторанний бізнес. У цій галузі успіх залежить від того, наскільки швидко та ефективно можуть бути оброблені замовлення клієнтів. Інформаційні технології дозволяють розв'язати цю проблему шляхом розробки спеціальних систем управління замовленнями.

Застосунок управління замовленнями “Replica” буде важливим кроком для закладу відпочинку у напрямку автоматизації та покращення обслуговування клієнтів. Це дозволить закладу виділитися на ринку та надасть нові інтерактивні можливості для потенційних клієнтів закладу. В першу чергу, це може бути зацікавленість в роботі системи та бажанні клієнта дізнатися як це працює в реальному житті. Клієнти зможуть замовляти страви та напої, переглядати меню, а також забронювати столик у зручний для них час. Застосунок буде працювати в онлайн-режимі, що дозволить клієнтам здійснювати замовлення з будь-якого місця, де є доступ до Інтернету. Також, застосунок буде підтримувати різні способи оплати, такі як оплата карткою, електронними грошима та інші. Проте дана можливість буде реалізована за допомогою сервісу посередника. Зберігання та обробка платіжних даних клієнтів велика відповідальність, тому краще покласти це завдання на сервіс, який орієнтований саме на такий тип послуг. Це значно спростить процес оплати для клієнтів та позитивно позначиться на їхньому досвіді від відвідування закладу. Крім того, наша система буде містити аналітичні інструменти, що дозволяють збирати та аналізувати дані про замовлення, найбільш популярні страви та напої, а також найбільш вигідні часи для роботи закладу. Це надасть можливість менеджменту зробити обґрунтовані рішення та покращити ефективність роботи закладу.

Основна мета розробника полягає в створенні веб застосунку у форматі WebAssembly зі зручним та інтуїтивно зрозумілим інтерфейсом для всіх користувачів, включаючи клієнтів та персонал закладу. Кожному користувачеві буде наданий відповідний інтерфейс згідно з його роллю в системі. Для досягнення мети планується покращити теоретичні знання, як про засоби Blazor WebAssembly, так і .NET загалом. А також покращити та розвинути нові, практичні навички роботи з технологією. Разом з цим, розробник ставить перед собою ціль, ознайомитися з іншими популярними технологіями, які вирішують низку питань в проєктуванні систем подібного типу. Що власне спростить сам процес розробки веб застосунку та зробить його більш надійним для кінцевого користувача, а разом з цим для власника закладу. Планується розглянути проблематику адміністрування закладів відпочинку та володіння особистою інформацією клієнтів, для забезпечення цілісності та надійності зберігання інформації, з метою запобігання крадіжки та розповсюдження даних.

Для досягнення поставленої мети, ми здійснили детальний аналіз ринку конкурентів. Виділили особливості та можливості кожного з них. Оглянули їхні системи та застосунки, які використовуються для інформування та власне взаємодії з клієнтом, в плані реклами чи надання послуг. Розробили план робіт та обрали стек технологій на базі якого буде розроблено наш програмний продукт. Визначили перелік завдань, які потрібно виконати для досягнення бажаних результатів:

- здійснити аналіз стану розробки систем управління закладами відпочинку;
- спроєктувати систему:
    - проєктування БД;
    - планування архітектури;
    - вибір технологій;
- розробити систему;

Об'єктом дослідження є система управління лаунж-баром.

Предметом дослідження процес адміністрування лаунж-бару та технології, які формуватимуть веб застосунок.

У даній роботі описано створення веб застосунку під назвою “Replica”. Це веб система управління замовленнями лаунж-бару. Замовник та розробник програмного продукту представлені в одній особі. Тема обрана через безпосередню причетність розробника до відкриття вищевказаного закладу відпочинку. Потреба в системі виникла через бажання організувати якісну роботу закладу, та впровадженні нових технологій у сферу адміністрування закладів відпочинку.

РОЗДІЛ 1
ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1. Опис предметного середовища

Технологічний прогрес не стоїть на місці та щодня надає розробникам нові можливості для проєктування та створення програмних продуктів. Щодня на ринку на ринку з’являються нові інструменти, платформи та бібліотеки. У програмістів з’явилася можливість створювати бібліотеки з власних рішень. Саме це посприяло розробці великої кількості бібліотек, які вирішують певні завдання і прискорюють розробку проєктів. Дедалі більше технологій розробляється з підтримкою кросплатформності, що збільшує можливості програміста та дає змогу створювати програми, які можуть працювати на різних операційних системах та пристроях.

Отже, можна зробити висновок, що аналогів у кожної технології є доволі багато. Тому розробник має вірно підібрати стек технологій для забезпечення найкращого результату, відповідно до вимог які встановлено до функцій розроблюваного продукту. Вибір технологій залежить від ряду факторів, таких як тип продукту, його масштабність, вимоги до продуктивності (швидкодії), доступні ресурси та уміння розробника. Під час вибору стека технологій також важливо враховувати майбутні перспективи та тренди в галузі розробки програмного забезпечення. Технології швидко змінюються, і те, що було популярним кілька років тому, може вважатися застарілим сьогодні. Тому розробник мусить відстежувати тенденції та перспективи розвитку технологій. А також потреба ринку в розробниках зі знанням певної технології, що власне й вказує на її популярність.

Система управління замовленнями лаунж-бару має бути доступною для потенційних користувачів у веббраузері. Тому важливо вибрати фреймворк на базі якого розроблятиметься проєкт “Replica”. А також середовище розробки та графічні редактори.

1.1.1. Вибір фреймворку

Фреймворк (англ. framework) — це програмна структура, призначена для створення та прискорення процесу розробки програмного продукту. Простіше дану структуру можна розглядати як комплекс бібліотек, при цьому він має ряд обмежень, які встановлюють правила написання коду та організації структури проєкту.

Головна ціль фреймворку, дати розробнику зручне середовище для програмування з великим та масштабованим функціоналом. Найчастіше фреймворки використовуються для створення веб додатків. Головне завдання фреймворку допомогти розробнику вибудувати правильну бізнес логіку та організувати робочий процес.

Починаючи розробку веб додатка, перед програмістом постає питання, який фреймворк краще використати. Вибір часто залежить від знань розробника та вміння працювати з технологією. Деякі популярні фреймворки веброзробки використовують різні мови програмування. Розглянемо кілька з них:

● Angular — один за найпопулярніших фреймворків для написання клієнтської частини вебдодатка, розроблений під керівництвом команди Angular Team (Google LLC). Базується на мовах програмування TypeScript та JavaScript, завдяки чому надає потужні інструменти для створення складних односторінкових додатків (Single Page Applications - SPA) з багатофункціональним інтерфейсом користувача (див. рис. 1.1).

Особливості Angular:

1. Компонентна архітектура : Angular реалізований на концепції компонентів, що дозволяє створювати повноцінні блоки коду з власними шаблонами, стилями та логікою. Компоненти доступні для повторного використання та комбінування для побудови складних інтерфейсів.

2. Двостороннє зв'язування даних: в Angular є механізм двостороннього зв'язування даних, що дозволяє автоматично синхронізувати дані між моделлю

та її представленням. Це спрощує роботу з формами та полегшує процес взаємодії з користувачем.

3. Впровадження залежностей: Angular використовує механізм ін'єкції залежностей для керування залежностями між компонентами та сервісами. Це робить код модульним та легкозмінним.

4. Розширений набір інструментів: Angular містить великий набір інструментів для розробки, таких як Angular CLI (Command Line Interface) для швидкого створення проєктів, компілятор шаблонів, модульна система та інші.

5. Підтримка TypeScript: Angular підтримує таку мову програмування як TypeScript, яка є розширенням JavaScript з реалізованою типізацією. TypeScript забезпечує більшу стабільність та розширюваність під час розробки.

6. Модульність та розширюваність: Angular дозволяє розділяти проєкт на окремі модулі, які за потреби можна замінити. Тобто дає змогу розділити програму на рівні для кращої організації коду. Це дозволяє легко розширювати функціональність додатка та підтримувати його масштабованість.

Рис. 1.1. Логотип скриптового фреймворку Angular.

Джерело: [https://www.html.it/guide/guida-angularjs].

Angular є потужним фреймворком для розробки вебдодатків, з великою спільнотою розробників від компанії Google LLC, що забезпечує доступ до багатьох ресурсів, документації та підтримки. Він надає зручність у розробці, швидкість та розширюваність проєктів. Проте, перед використанням Angular розробник має оцінити свої потреби, рівень знань та вимоги до проєкту, щоб зробити правильний вибір фреймворку.

● React — це одна з найпопулярніших JavaScript бібліотек для розробки користувацьких інтерфейсів. Не являється фреймворком, проте в парі з іншими сумісними бібліотеками реалізовує ті можливості, які притаманні фреймворку. Вона розроблена командою Facebook і використовується для побудови ефективних та масштабованих вебдодатків (див. рис. 1.2).

Особливості React:

1. Компонентна архітектура: React побудований на концепції компонентів, що дозволяє розділяти користувацький інтерфейс на незалежні, повторно використовувані частини. Це спрощує розробку, тестування та підтримку коду.
2. Віртуальний DOM: React використовує віртуальний DOM для ефективного оновлення та маніпулювання сторінкою. Зміни відбуваються спочатку в віртуальному DOM, а потім React автоматично оновлює лише змінені елементи на сторінці. Це призводить до поліпшення продуктивності додатків.
3. Односторінкові додатки: React може використовуватись для створення односторінкових додатків (SPA), де зміна контенту відбувається без перезавантаження сторінки. Це забезпечує швидку та зручну взаємодію з користувачем.
4. JSX: JSX є розширенням синтаксису JavaScript, яке дозволяє використовувати HTML-подібні теги для побудови компонентів у React. Це спрощує створення шаблонів та забезпечує більш зрозумілий код.

5. Розширюваність: React має багато розширень (бібліотек та плагінів), які допомагають розширити його функціональність та забезпечити додаткові можливості, такі як маршрутизація (React Router) або керування станом (Redux, MobX).

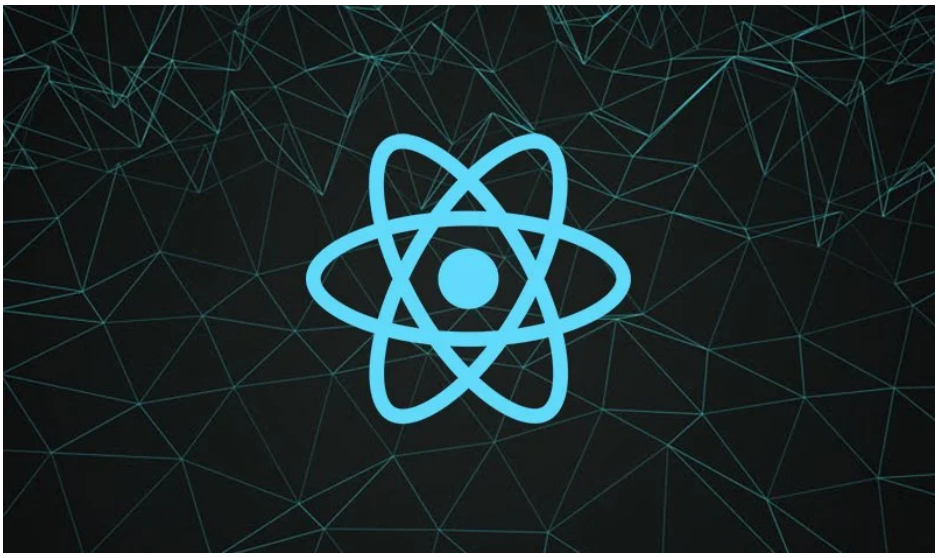

Рис. 1.2. Логотип JS бібліотеки React.

Джерело: [https://www.trio.dev/blog/react-is-used-for].

React також має широку спільноту розробників, що забезпечує багато ресурсів, документацію та підтримку. Він часто використовується для створення інтерфейсів користувача в різних сферах, включаючи вебдодатки, мобільні додатки та навіть настільні застосунки.

● Blazor WebAssembly (Blazor WASM) - це фреймворк розробки вебдодатків, розроблений командою Microsoft. Основна особливість Blazor WASM полягає в тому, що він дозволяє розробляти вебдодатки, використовуючи мову програмування C# без необхідності використовувати JavaScript (див. рис. 1.3).

Особливості Blazor Wasm:

1. Вебдодатки на стороні клієнта: Blazor Wasm дозволяє створювати вебдодатки, які виконуються на стороні клієнта, у веббраузері. Це означає, що весь код

виконується безпосередньо в браузері без необхідності звертатися до сервера для виконання операцій.

2. Використання C#: Завдяки Blazor Wasm розробники можуть використовувати мову програмування C# для створення вебдодатків. Це дозволяє розробникам, які знайомі з C# і .NET, перенести свої навички на веброзробку без необхідності вивчати JavaScript.

3. Використання .NET: Blazor Wasm побудований на базі платформи .NET і використовує .NET runtime в браузері. Це дозволяє використовувати багато знайомих і потужних бібліотек та інструментів з платформи .NET для розробки вебдодатків.

4. Компонентна модель: Blazor Wasm використовує компонентну модель, подібну до Angular або React. Розробники можуть створювати компоненти з кодом C# та розміткою, яка описує, як компонент повинен бути відображений на екрані. Компоненти можуть бути повторно використані і складатися для створення складних інтерфейсів користувача.

5. Двостороннє зв'язування даних: Blazor Wasm підтримує двостороннє зв'язування даних, що дозволяє автоматично оновлювати інтерфейс користувача при зміні даних, і навпаки. Це полегшує роботу з даними та їх відображенням у вебдодатку.

6. Розробка одночасно для front-end та back-end: Blazor WASM дозволяє розробляти повністю функціональні вебдодатки, використовуючи C# як для клієнтського, так і для серверного коду. Це спрощує розробку та підтримку додатків, оскільки розробники можуть використовувати спільні бібліотеки та компоненти для обох частин додатка.

Рис. 1.3. Логотип фреймворку Blazor.

Джерело: [https://www.editions-eni.fr/blog/blazor-questions-a-notre-expert].

Blazor Wasm надає розробникам можливість використовувати знайомі інструменти та мову програмування для розробки вебдодатків без необхідності вивчати JavaScript. Однак, слід враховувати, що Blazor Wasm є новітнім фреймворком і має свої особливості та обмеження, які варто враховувати при виборі фреймворку для конкретного проєкту.

Після детального огляду та визначення особливостей кожного фреймворку, було прийнято рішення використати Blazor. Головна причина такого вибору, можливість використовувати для розробки мову програмування C# як для серверної, так і для клієнтської частини проєкту. А також власна зацікавленість в технології WebAssembly. Адже WASM забезпечує високу продуктивність, близьку до продуктивності нативного коду (машинний код, який виконується безпосередньо процесором цільової апаратної платформи). Це досягається завдяки компіляції байт-коду WASM безпосередньо в машинний код. Завдяки чому, програми написані на мовах, які компілюються в WebAssembly, виконуються швидше ніж програми написані на скриптових мовах програмування таких як JavaScript. Також дана технологія розроблена з урахуванням безпеки, код програми запускається в

сандбоксі браузера (англ. browser sandbox), це ізольоване середовище яке захищає застосунок від потенційно шкідливого коду який міг бути завантажений з мережі.

1.1.2. Вибір середовища розробки

Вибір середовища розробки (англ. Integrated Development Environment, скорочено IDE) також є важливим, адже у кожного є свої особливості. Проте у нашому випадку, з вибором буде простіше. Оскільки фреймворк Blazor розроблений компанією Microsoft, доцільно використовувати середовище розробки Visual Studio, розроблене цією ж компанією. Я вважаю, що вибір Visual Studio для роботи з фреймворком Blazor є найкращим варіантом, адже це потужний інструмент для розробки програмного забезпечення, який надає широкі можливості для програміста.

Проте це не єдине IDE, яке планується використовувати під час розробки проєкту “Replica”. Оскільки, система управління замовленнями лаунж-бару, має бути реалізована як веб застосунок, для її розробки буде використано такі технології як HTML та CSS. Для зручності роботи з даними технологіями, на мою думку, доцільно використовувати інше IDE, таке як Visual Studio Code. Він орієнтований на роботу з даними технологіями та має ряд розширень, які зроблять зручним проєктування скелета вебсторінок. Я вважаю, що розробити зовнішній вигляд компонента значно зручніше використовуючи Visual Studio Code, після цього його з легкістю можна перенести в основний проєкт та модифікувати для роботи в основному коді програми.

1.1.3. Вибір середовища створення та обробки графічних матеріалів

Як середовища створення та обробки графічних матеріалів, було обрано Adobe Photoshop, Adobe Illustrator та Adobe Lightroom. Було прийнято рішення не використовувати аналоги даних програм, адже потрібно вивчати їх інтерфейс та пристосовувати до роботи з ними. Тому доцільно використовувати графічні

редактори з якими наявний досвід роботи. Я вважаю, що цих програм буде достатньо для того, щоб обробляти та створювати графічний матеріал для проєкту “Replica”. В основному буде використовуватися Adobe Photoshop та Adobe Lightroom. Ці програми надають потужний функціонал для обробки зображень, що дасть змогу обробити весь графічний матеріал відповідно до стилю користувацького інтерфейсу. Програму Adobe Illustrator буде використано лише для створення логотипа програмного продукту.

1.2. Огляд наявних аналогів

Зважаючи на те, що ринок програмного забезпечення для управління лаунж-барами не є дуже широким, аналіз можна провести у зведеному вигляді. Ось декілька систем управління лаунж-барами, які доступні на ринку:

- Lightspeed POS: це програмне забезпечення для керування бізнесом, яке дозволяє керувати запасами, замовленнями, фінансами та звітністю. Lightspeed POS пропонує різноманітні функції, такі як керування знижками та промоакція, збір даних про клієнтів та аналіз їх поведінки. Він також інтегрується з різноманітними платіжними системами, що дозволяє легко приймати платежі від клієнтів.

- Toast: це програмне забезпечення для керування ресторанним бізнесом, яке дозволяє керувати замовленнями, запасами, фінансами та звітністю. Toast має такі функції, як управління столиками, збір даних про клієнтів та аналіз їх поведінки. Він також інтегрується з різноманітними платіжними системами, що дозволяє легко приймати платежі від клієнтів.

- Revel Systems: це програмне забезпечення для керування ресторанним бізнесом, яке дозволяє керувати замовленнями, запасами, фінансами та звітністю. Revel Systems має різноманітні функції, такі як управління замовленнями зі знижками та промоакціями, збір даних про клієнтів та аналіз їх поведінки. Він також

інтегрується з різноманітними платіжними системами, що дозволяє легко приймати платежі від клієнтів.

- ShopKeep: це програмне забезпечення для управління роздрібним бізнесом, яке дозволяє керувати запасами, замовленнями, фінансами та звітністю. ShopKeep має такі функції, як управління знижками та промоакціями, збір даних про клієнтів та аналіз їх поведінки. Крім того, він дозволяє керувати програмою лояльності та робити звіти про продажі. Він також інтегрується з різноманітними платіжними системами, що дозволяє легко приймати платежі від клієнтів.

- Upserve: це програмне забезпечення для управління ресторанним бізнесом, яке дозволяє керувати замовленнями, запасами, фінансами та звітністю. Upserve має різноманітні функції, такі як управління меню, збір даних про клієнтів та аналіз їх поведінки. Він також дозволяє керувати програмою лояльності та робити звіти про продажі. Він інтегрується з різноманітними платіжними системами, що дозволяє легко приймати платежі від клієнтів.

- Square for Restaurants: це програмне забезпечення для керування ресторанним бізнесом, яке дозволяє керувати замовленнями, запасами, фінансами та звітністю. Square for Restaurants має такі функції, як управління меню, керування замовленнями та збір даних про клієнтів. Він інтегрується з різноманітними платіжними системами, що дозволяє легко приймати платежі від клієнтів. Крім того, він дозволяє керувати програмою лояльності та робити звіти про продажі.

Аналізуючи сайти потенційних конкурентів закладу, зробив висновок, що подібні системи на ринку України відсутні. Більшість закладів мають лише сторінки в соціальних мережах, де представлено адресу закладу та/або контактні дані. Частина лаунж-барів має власні сайти де реалізована можливість бронювання столика, за допомогою формування відповідної заявки на сайті. Підтвердження здійснюється телефонним дзвінком. Проте, переважна кількість сайтів має більш інформативний характер, особливого функціонала на них не виявлено.

Для поглибленого аналізу, було виділено кілька вебсайтів:

- Black Shisha: сайт даного закладу містить більш інформаційний характер. На ньому висвітлено коротку інформацію про заклад та його персонал. Наявні контактні дані, завдяки яким можна отримати консультацію, або здійснити бронювання місця. Також на сайті представлено меню лаунж-бару з цінами на кожну страву/напій та категорія подій які плануються, або вже відбулися в межах закладу. Особливого функціонала не виявлено.

  Сайт лаунж-бару «Black Shisha»: [https://blackshisha.com.ua/].

- B-hush: сайт даного закладу містить переважно інформаційний характер. Є можливість залишити заявку на бронювання місць за допомогою заповнення відповідної форми на сайті. Після чого менеджер оголосить клієнту можливі варіанти в телефонному дзвінку. Також висвітлено інформацію про події, які плануються, або вже відбулися в закладі. Є меню закладу, але представлене воно у вигляді файлів pdf формату, які клієнт може завантажити собі для ознайомлення. Додаткового функціонала не виявлено.

  Сайт лаунж-бару «b-hush»: [https://b-hush.com.ua/].

- Fiji: вебзастосунок даного закладу, має найбільшу подібність до системи, яку я маю намір розробити. Тут присутня можливість залишити заявку на бронювання місць в закладі, заповнивши відповідну форму, в якій користувач може додати час та дату бронювання, а також залишити коментар. Система надає можливість здійснити замовлення доставлення на певний асортимент страв від закладу. Оплата замовлення при отриманні. Також даний лаунж-бар має власний мобільний додаток, доступний на iOS та Android, основними функціями якого є система бонусів та висвітлення новин (див. рис. 1.4). Здійснити замовлення, або залишити заявку на бронювання в додатку неможливо.

  Сайт лаунж-бару «Fiji»: [https://fiji-loungebar.com.ua/].

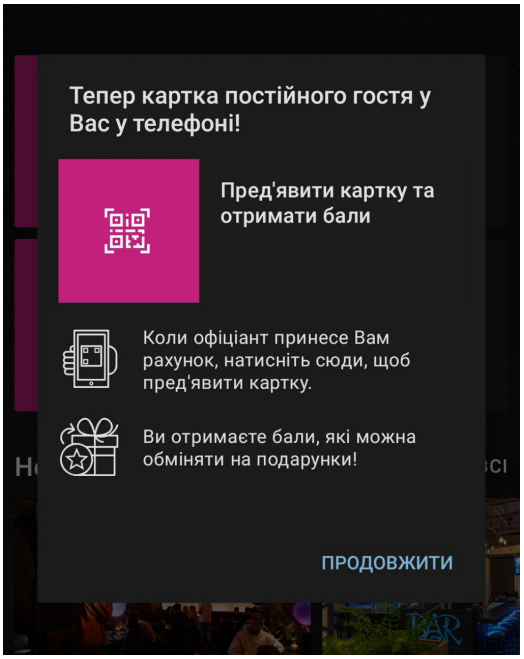

Рис. 1.4. Інтерфейс мобільного додатку лаунж-бару «Fiji».

Джерело: [Мобільний додаток лаунж-бару «Fiji»].

Отже, після аналізу інформації та детального ознайомлення з компаніями, які пропонують послуги з розробки (адаптації) програмного забезпечення під той чи інший варіант готельно-ресторанного бізнесу, в нашому випадку лаунж-бару, було прийнято рішення про розробку системи управління замовленнями. Система буде орієнтована саме під потреби лаунж-бару, це одна з причин відмінності з програмними продуктами які пропонує ринок. Також дана система в першу чергу орієнтована на взаємодію з клієнтом і лише після цього на персонал. Оцінивши ринок України в даному сегменті ринку, можна дійти до висновку, що подібні системи не використовуються, або не мають аналогів. Більшість сайтів мають інформативний характер і надають клієнтам контактну інформацію, в окремих випадках електронне меню. Бронювання місць в закладах реалізоване у вигляді форми, яку клієнт має заповнити контактною інформацією для зворотного зв'язку. Власне, це ще одна причина для розробки подібної системи, потреба в наданні користувачу альтернативного методу бронювання місць в закладі з метою уникнення телефонних дзвінків. При цьому ефективність даної послуги зросте, оскільки записи

про резервування будуть зберігатися в базі даних, що гарантує точність виконання даної послуги.

1.3. Постановка задачі

Для якісного управління замовленнями лаунж-бару, необхідно розробити систему у вигляді WebAssembly, завдяки якій буде здійснено контроль даних процесів. В першу чергу бронювання місць в закладі та здійснення замовлень. Додатково це адміністрування роботи закладу та збір статистичних даних для визначення важливих показників в роботі лаунж-бару. Оскільки, статистичні дані можуть інформувати власника про стабільність роботи закладу, популярність страв та напоїв, а також кількість відвідувачів відносно часу доби та днів тижня. Проте це додатковий функціонал, який може бути реалізованим за потреби.

Розроблювана система в першу чергу орієнтована на клієнтів закладу, тому в процесі розробки потрібно зосередитися саме на цьому. Та наділити програму всіма необхідними візуальними та функціональними властивостями, які будуть зрозумілими кінцевому користувачеві. Мета розробки такої системи полягає в автоматизації процесу формування замовлень.

Клієнтська частина проєкту є важливою складовою для забезпечення взаємодії між користувачем та закладом в обличчі персоналу лаунж-бару. Адже саме правильно, а головне якісно створений користувацький інтерфейс є гарантованою запорукою успіху розроблюваної системи. Від зручності та візуальної складової проєкту, залежить його популярність, а разом з цим потреба в даній системі. До клієнтської частини програмного продукту, було висунуто ряд вимог:

1. Реєстрація та авторизація: кінцевий користувач мусить зареєструватися в системі для того, щоб користуватися послугами закладу віддалено та в його межах. На основі обробки персональних даних, які надає користувач в процесі реєстрації, будуть формуватися його замовлення. Це необхідно для ідентифікації клієнта. Якщо користувач не може та/або не має бажання

проходити реєстрацію, замовлення можна буде здійснити в анонімному режимі з допомоги персоналу лаунж-бару в його межах.

2. Доступ до меню: як анонімний, так і авторизований користувач має мати можливість переглядати меню закладу, як в цілях формування замовлень, так і для власного ознайомлення.

3. Здійснення замовлень, бронювань: авторизований в системі користувач має мати можливість здійснити замовлення товарів та послуг лаунж-бару дистанційно. Неавторизований користувач матиме можливість замовити страви лише в межах закладу.

4. Безпека особистих даних користувача: інформація кожного клієнта має бути збережена і захищена від пошкоджень та крадіжки. Для цього програмний продукт має зберігати пароль користувача в зашифрованому вигляді. При цьому варіанті навіть якщо дані буде вкрадено, авторизуватися за допомогою цих даних зловмисник не зможе. Всі паролі будуть збережені у базі даних з використанням алгоритму одностороннього шифрування.

5. Зручний користувацький інтерфейс: інтерфейс користувача має бути простим для розуміння та легким у користуванні. Стиль сайту має виглядати молодіжним, адже такі заклади як лаунж-бар орієнтовані в першу чергу на клієнтів молодшого віку.

Клієнтська частина надає інтерфейс для користувачів, проте обробка запитів та власне робота з даними виконується на серверній частині проєкту. Тому також важливо розробити та організувати правильну роботу серверної частини проєкту. Використати всі необхідні технології та засоби розробки, для забезпечення стабільної та безвідмовної роботи сервера. Тому, також було висунуто вимоги до серверної частини проєкту:

1. Реалізація правильної архітектури: для того, щоб забезпечити якісну роботу застосунку, рекомендується використовувати розділену архітектуру з чітким

розділенням логіки серверної частини від клієнтської. Правильне архітектурне рішення дозволить легко розширювати систему в майбутньому. Це означає, що можна додавати нові модулі та функціонал без потреби в переписуванні значної частини коду. Так звана модульність, коли програма складається з декількох підпроєктів.

2. Комунікація з клієнтською частиною: для взаємодії клієнтської та серверної частини потрібно використати один з найпоширеніших протоколів передачі даних, а саме протокол передачі гіпертексту (англ. Hypertext Transfer Protocol, скорочено HTTP). Клієнтська частина, у нашому випадку Blazor WASM може відправляти різні типи запитів HTTP до серверної частини, такі як GET, POST, PUT, DELETE. Кожен тип запиту має свою специфіку і використовується для різних цілей. Наприклад, запит GET використовується для отримання ресурсів з сервера, тоді як POST використовується для надсилання даних на сервер. Самі ж запити реалізовані у вигляді URL покликань, до яких звертається клієнтська частина.

3. Зберігання даних: потрібно обрати технологію для зберігання даних, щоб вся необхідна для користувача інформація була доступною в реальному часі. Для цього потрібно реалізувати базу даних, спроєктувати її архітектуру. Доцільно використати MySQL, NoSQL або MongoDB.

4. Безпека: використання технологій авторизації та автентифікації користувача, для забезпечення безпеки даних та конфіденційності. Доцільно використати технологію JWT (JSON Web Tokens) для безпечного обміну інформацією між клієнтською та серверною частинами проєкту.

5. Управління помилками: необхідно реалізувати механізми для обробки помилок та повідомлень про помилки на серверній стороні. Використати зручність інструментів, таких як middleware або фільтри, для перехоплення та обробки помилок.

Висновки до розділу

В процесі роботи, ми здійснили аналіз предметного середовища. Проаналізували популярні фреймворки для розробки вебзастосунків, виділили особливості та переваги кожного з них. Навели детальний опис особливостей використання даних фреймворків, та прийняли рішення використати для розробки системи управління замовленнями лаунж-бару “Replica” технологію розроблену компанією Microsoft, а саме Blazor WebAssembly. Адже враховуючи переваги які надає WASM, системі буде гарантовано швидкодію, та захист від шкідливого коду. Технологія Blazor є мультиплатформною і гарантує стабільну роботу в незалежності від операційної системи пристрою на якому буде запущено наш вебзастосунок.

Також, нами було обрано для використання два IDE розроблених компанією Microsoft. Це Visual Studio та Visual Studio Code. Вони містять всі необхідні інструменти для розробки проєкту з використанням фреймворку Blazor WASM.

Visual Studio є повнофункціональним IDE, яке надає розширені можливості для розробки .NET-додатків. Воно забезпечує багато інструментів для роботи з Blazor WASM, включаючи шаблони проєкт, підтримку інтелектуального кодування, налагоджувальний інтерфейс, вбудовані середовища керування пакетами та інші корисні функції.

Visual Studio Code це потужний редактор коду, який надає всі необхідні функції для роботи з технологіями HTML та CSS. Тому доцільно використовувати його для створення структури компонентів. Значно зручніше ніж реалізовувати компоненти у середовищі Visual Studio. Також в ньому доступна велика кількість плагінів, які можуть полегшити процес розробки.

Для обробки графічних матеріалів ми обрали програми запропоновані компанією Adobe. Функціоналу даних застосунків буде достатньо для задоволення всіх потреб в плані роботи з графічним матеріалом для системи “Replica”.

Ми встановили вимоги до клієнтської та серверної частини проєкту, виконання всіх вимог гарантує якісний результат. Система отримає всі функціональні

особливості які необхідні для стабільності та швидкодії програмного продукту. При цьому всі частини проєкту будуть незалежними та при необхідності можуть бути заміненими на інші.

РОЗДІЛ 2
ІНФОРМАЦІЙНЕ ТА МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

2.1. Аналіз предметної області

Використання інформаційних технологій у сферах послух не є новизною, а навпаки є популярним трендом протягом останнього десятиліття. Банки, магазини, служби доставлення та багато інших закладів та підприємств які надають послуги клієнтам намагаються розробити зручні застосунки для взаємодії з користувачем. Візьмемо до уваги мобільні додатки та вебсайти банків. Там реалізований весь необхідний функціонал для взаємодії з користувачем, тепер не потрібно відвідувати відділення банку для отримання послуг, можна скористатися мобільним додатком, щоб отримати інформацію про стан рахунку, платіжної історії та багатьох інших послуг банку. З недавніх пор деякі з них навіть надають послуги з відкриття рахунку за допомогою мобільного застосунку.

Процес популяризації технологій які автоматизують певні буденні справи кожного з нас вже не вийде спинити, з часом подібні програми з’являться у всіх можливих сферах нашого життя. Тому й сфера готельно-ресторанного бізнесу не виняток, система управління замовленнями лаунж-бару це крок до автоматизації процесу замовлення та бронювання місць в закладі відпочинку. Важливість розробки такого проєкту як “Replica” говорить сама за себе. Системи подібного типу розроблюються вже давно, як може бути зрозуміло з дослідження проведеного в першому розділі кваліфікаційної роботи, проте вони орієнтовані на персонал закладу. Ці рішення дозволяють вести контроль за замовленнями та їх оплатою, збирають та виводять статистичні дані про дохід, відвідування та популярність певного типу страв. Проте не містять інтерфейсу який може використати клієнт закладу. Також деякі заклади використовують вебсайти (зазвичай односторінкові) або соціальні мережі для поширення інформації про заходи та контактні дані адміністраторів закладу.

Досліджуваним об'єктом є процеси як відбуваються в закладах відпочинку, а саме лаунж-барах під час: оформлення замовлення, бронювання місць в закладі, оплати замовлення, встановлення вікових обмежень, а також процесів підтвердження, обробки та завершення виконання замовлення. Також доцільно дослідити процес збирання та обробки особистої інформації клієнтів, забезпечення цілісності та конфіденційності наданих користувачем в процесі реєстрації та використання програмного продукту даних. Використати для цього всі необхідні заходи безпеки, методи та технології шифрування, накопичення та зберігання даних.

2.1.1. Аналіз процесів формування та обробки замовлень

Аналіз процесів формування замовлення є важливим етапом в розробці системи управління замовленнями лаунж-бару. Адже для формування правильних зв'язків та встановлення вимог до інформації, яку повинен надати клієнт для системи, щоб формування замовлення стало можливим та в першу черву вірним, потрібно дослідити як саме відбувається процес замовлення.

Проєкт “Replica” мусить бути реалізовано з відкритим для користувача (клієнта закладу) інтерфейсом. Для цього потрібно встановити ряд вимог до особистої інформації, яку має надати клієнт для закладу, щоб надалі реалізувати процес замовлення. Виділимо головний тип користувачів системи, в нашому випадку це потенційний клієнт закладу, тобто особа яка має на меті отримати послуги закладу для задоволення особистих потреб. В межах закладу для формування замовлень, без використання системи управління замовленнями, особисту інформацію клієнт не надає. Зазвичай йде прив’язка замовлення до номера стола в залі закладу. Проте, для підтвердження замовлення яке здійснено через систему, потрібно ідентифікувати користувача, щоб працівники закладу могли бути впевненими, що замовлення або ж бронювання місця належить саме цьому клієнту.

2.1.1.1. Особиста інформація клієнтів

Для ідентифікації клієнта в процесі реєстрації в застосунку потрібно зібрати необхідні дані. На основі них буде сформовано обліковий запис користувача, завдяки чому з’явиться змога зв'язувати інформацію про замовлення з користувачем який здійснив його. Зазвичай сервіси збирають наступну інформацію:

1. ПІБ клієнта
2. Контактний номер телефону
3. Контактна електронна пошта
4. Дата народження

Проте, в нашому випадку цього буде недостатньо, необхідно розширити список з інформації яку має надати клієнт для системи “Replica”. Після детального аналізу, було додано додаткові вимоги до особистої інформації користувача:

1. Місто: можливо лаунж-бар в майбутньому перетвориться в мережу закладів і це потрібно врахувати на етапі аналізу. Завдяки інформації про місто можна здійснювати прив'язку користувачів до певного закладу та надсилати їм актуальну інформацію, про заходи та події, які відбуваються в найближчому до них лаунж-барі.
2. Ім’я користувача (ІК): так склалися обставити, що користувачі можуть мати однаковий ПІБ, тому доцільно додатково зберігати ім’я користувача яке буде унікальним для кожного клієнта. ПІБ використовуватиметься для звернення персоналу до клієнта, ім’я користувача для підтвердження. Також в майбутньому ІК може бути використано в додатковому функціоналі системи, наприклад під час створення коментарів клієнтом та інших.
3. Зображення: також користувач за бажанням може завантажити особисте фото, або картинку для відображення в профілі користувача.

Даної інформації буде достатньо для того, щоб працівники закладу могли взаємодіяти з клієнтами використовуючи систему “Replica”.

2.1.1.2. Інформація про замовлення

Зазвичай в лаунж-барах інформацію про замовлення надає клієнт безпосередньо для офіціанта або бармена. Про це створюють запис в блокноті або ж системі, яку використовує заклад для адміністрування. Зазвичай там міститься коротка інформація про страви та напої, а також особисті побажання клієнта. Орієнтовно даний процес виглядає наступним чином (див. Рис. 2.1).

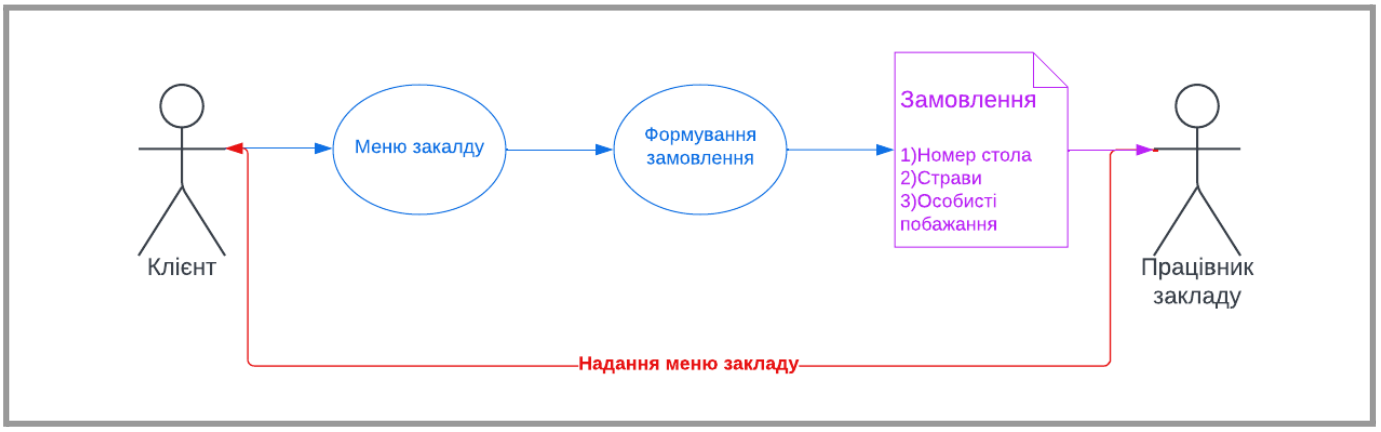

Рис. 2.1. Діаграма процесу формування замовлення.

Джерело: [Створено автором].

Проте, проєкт “Replica” має отримувати інформацію від користувача в іншому вигляді. Послуги закладу, а точніше страви, напої, кальяни та інше мають бути доступні як сутності в системі, з якими буде безпосередньо взаємодіяти користувач. Тобто інформація про замовлення не буде отримана від користувача в вигляді інформаційного поля, адже користувач буде обирати вміст свого замовлення безпосередньо в системі. Процес формування замовлення в проєкті “Replica” (див. Рис. 2.2).

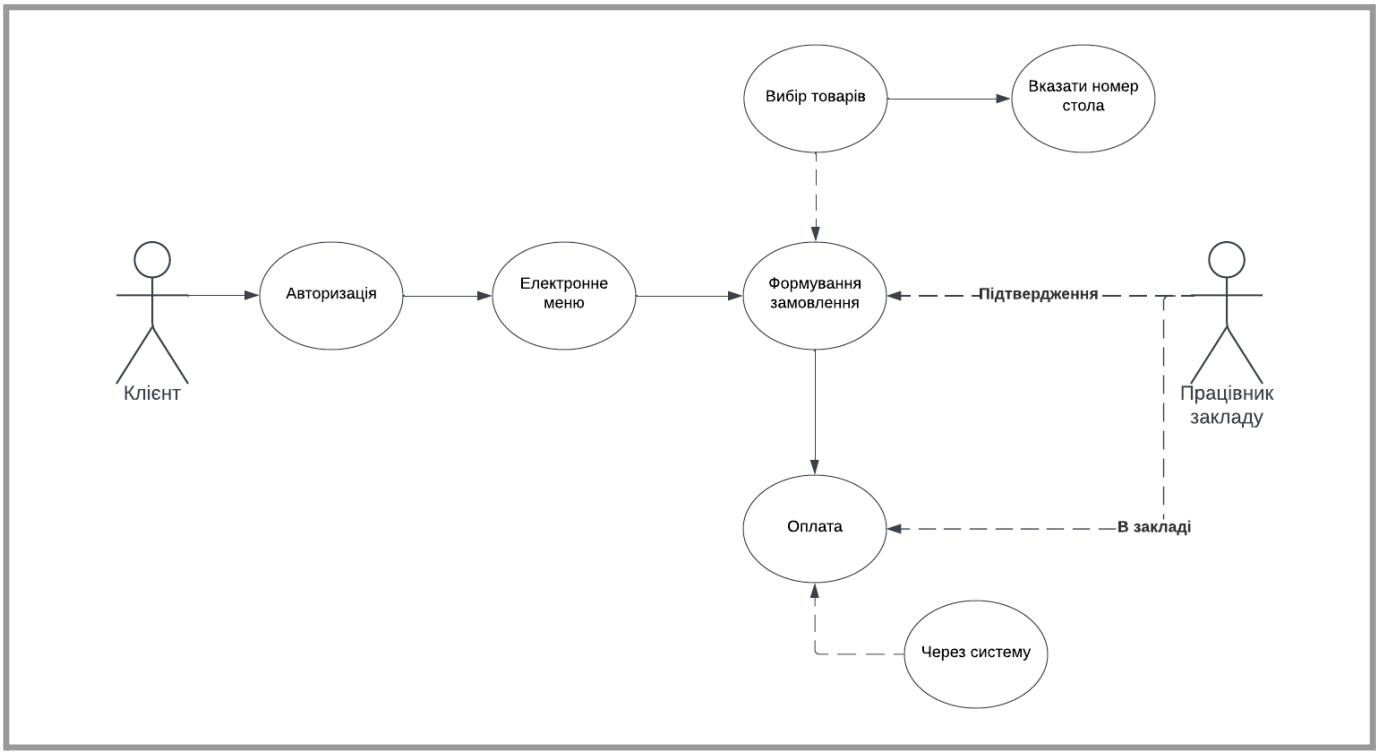

Рис. 2.2. Діаграма процесу формування замовлення з використанням системи “Replica”.

Джерело: [Створено автором].

На основі діаграми наведеної вище можна здійснити опис інформації, яку буде використовувати система для формування замовлень:

1. Ідентифікатор користувача: для створення зв’язку між замовленням та користувачем, буде використано ідентифікатор користувача, який створює система автоматично. В межах системи, використання унікального ідентифікатора буде достатньо для формування зв'язку.

2. Список з ідентифікаторів товарів: вказуючи слово товар, мається на увазі страви, напої, кальяни та інші послуги що надаються лаунж-баром. Оскільки для формування замовлень буде використовуватися система, достатньо передавати список з ідентифікаторів товарів замовлених клієнтом.

3. Коментар до замовлення: це додаткова інформація яку надає користувач, особисті вподобання що до вмісту страви або додаткові прохання.

4. Номер стола: для завершення потрібно вказати номер стола за яким наразі знаходиться користувач. Працівники закладу мають розмістити номера на кожному столі для того, щоб клієнт знав номер стола і міг вказати його під час формування замовлення.

Якщо клієнт не захоче реєструватися в системі, закладом має бути передбачене оформлення замовлення на анонімного користувача. Формувати таке замовлення мусить персонал лаунж-бару. Меню для клієнта має надаватися в паперовому вигляді, або ж за рішенням керівництва закладу для клієнта може бути запропоновано перегляд меню з використанням електронних пристроїв закладу.

2.1.1.3. Підтвердження замовлення

В лаунж-барах замовлення підтверджуються уточненням працівника закладу, який робить запис замовлення. Це власне й можна назвати підтвердженням замовлення. Проте в системі “Replica” потрібно реалізувати відповідний функціонал для підтвердження замовлення. В першу чергу клієнт має отримати сповіщення про те, що його замовлення було підтверджено і персонал закладу почав його виконувати. Власне, для персоналу також зручно коли замовлення має поле статусу, завдяки цьому можна розділити замовлення на категорії: очікується підтвердження, підтверджено, виконано та відхилено.

Також потрібно врахувати той факт, що замовлення може потребувати внесення певних змін. Тобто, можливо з певних причин приготування страви або коктейлю стане неможливим (скінчилися потрібні інгредієнти). Також можливо, що доданий до замовлення коментар не можливо зрозуміти або він потребує уточнення від клієнта.

Для клієнтів які не мають змоги, або бажання використовувати систему “Replica” для здійснення замовлень, заклад має передбачати виконання даного процесу без використання програми самотужки. З запропонованих це підтвердження замовлення у звичайному режимі, завдяки взаємодії клієнта та безпосередньо відповідального за це працівника лаунж-бару.

2.1.2. Обмеження вхідних та вихідних даних

Обмеження вхідних та вихідних даних стало важливою практикою в розробці програмного забезпечення для забезпечення стабільної та надійної роботи системи. Розроблювана нами система не є виключенням та потребує встановлення обмежень на вхідні та вихідні дані. Основною метою встановлення обмеження даних це запобігти ін'єкції шкідливого коду, отримання несанкціонованого доступу та використання функціонала, який непризначений для даного користувача.

Розглянемо декілька загальних принципів обмеження вхідних та вихідних даних:

1. Валідація вхідних даних: першим етапом роботи з даним є перевірка їх на наявність некоректних або ж навіть небезпечних значень. Це може включати перевірку формату, типу та допустимих значень. Використання валідації допомагає запобігти вразливостям, таким як SQL-ін'єкції або впровадження зловмисного коду в систему.

2. Екранізація та екранування вихідних даних: перед виведенням або передачею даних користувачеві, їх потрібно переконатися, що вони не містять небезпечних символів або коду, який може бути виконаний. Може включати екранування спеціальних символів, використання безпечних методів форматування даних (валідація).

3. Обмеження доступу до даних: доступ до даних системи має бути обмежений та поділений на рівні доступу(використання ролей користувачів в системі). Потрібно використовувати принцип обмеження дозволеного, де користувачеві надаються лише необхідні права для взаємодії з певним функціоналом.

4. Захист від переповнень буфера: під час обробки вхідних даних потрібно встановити перевірку, щоб уникнути можливих переповнень буфера, які можуть призвести до об'ємних витрат пам'яті або виконання шкідливого коду.

5. Використання параметризованих запитів: для взаємодії з базою даних потрібно використовувати параметризовані запити, щоб уникнути SQL-ін'єкцій, які можуть надати несанкціонований доступ, модифікацію даних або ж повне її знищення.

6. Використання безпечних протоколів і шифрування: для передачі даних через мережу обов’язковим є використання безпечних протоколів передачі даних, таких як HTTPS, вони забезпечують шифрування трафіку та захист від прослуховування (перехоплення).

Власне на основі цього можна зробити висновки, що при використанні наведених вище рекомендацій для обмеження вхідних та вихідних даних, можна забезпечити стабільну роботу застосунку з захистом від аномальних даних та ін’єкцій коду. Важливо ретельно перевіряти та обмежувати всі дані, які вводяться або виводяться з системи, та гарантувати, що вони відповідають очікуваному формату та валідності.

Для проєкту “Replica” в першу чергу буде реалізовано валідацію введених користувачем даних та обмеження доступу до даних з використанням системи ролів. Завдяки цьому гарантується, що до бази даних буде записано лише перевірені та правильні дані. А також кожен користувач отримуватиме доступ лише до того рівня інформації який йому необхідний для роботи у системі.

2.2. Проєктування системи

Для правильного підбору архітектури та стеку технологій необхідно провести аналіз того, які вимоги поставлені до проєкту. На основі цього, можна підібрати рішення яке задовольнить потреби повністю, або частково. Перевага програмування в тому, що ми можемо використати кращі сторони кожної технології та об’єднати їх в єдину систему, для отримання бажаного результату. А правильно підібрана архітектура зробить наш продукт легко розширюваним та простим для розуміння

іншим розробникам які хоча б раз працювали з архітектурними рішеннями. Адже всі архітектурні патерни подібні між собою і дають відповідь на питання, де знаходиться той чи інший компонент.

2.2.1. Архітектурне рішення

Першим етапом створення якісного програмного продукту, є вибір архітектурного рішення. Архітектура програми — це структурна карта програмного забезпечення, завдяки якій здійснюється поділ застосунку на рівні. У ній детально описано, як кожна одиниця програмного забезпечення пов’язана одна з одною, та як вони взаємодіють, щоб задовольнити вимоги подальшого користувача. Тобто, в результаті розробки, ми отримуємо програмний продукт поділений на серію рівнів. Завдяки яким забезпечується краще функціонування та розуміння того, як працює програма, та до чого призводять зміни, в одному з її шарів.

Протягом останніх років, була запропонована низка ідей, щодо архітектурних рішень. До них належать:

- Hexagonal Architecture (a.k.a. Ports and Adapters) Алістера Кокберна.
- Onion Architecture Джеффрі Палермо.
- Screaming Architecture Роберт С. Мартін.
- MVC (Model-View-Controller) Трюгве Реєнскауга.
- DCI (Data-Context-Interaction) від Джеймса Коплієна та Тригве Реенскауга.
- BCE (Boundary-Control-Entity) Івара Якобсона.
- Clean Architecture Роберт С. Мартін.

Насправді всі рішення є подібними між собою і нагадують структуру одне одного. Найпоширенішими рівнями архітектур є:

- Рівень презентації — має справу з користувацьким інтерфейсом, тобто тією частиною програми, яка обробляє вхідні дані користувача, керує запитами які здійснює користувач, передає їх до сервісів даних, представляє вихідні дані, та має справу з усіма іншими формами взаємодії між користувачем і додатком.

- Рівень обслуговування даних — виступає мостом між рівнем презентації та рівнем бізнес-логіки. З точки зору безпеки, це стіна, яка відокремлює те, що робить користувач, від основної логіки додатка, роблячи його більш безпечним.

- Рівень бізнес-логіки — операційний центр, на цьому рівні відбувається обмін або обробка даних, кодування введених користувачем даних та/або підготовка інформації для передачі на рівень представлення. Наприклад, у динамічному вебдодатку це та частина програми, яка вирішує, яка інформація потрібна для рівня представлення. Вона бере інформацію зі сховища даних, виконує необхідну підготовку та надсилає її для відображення користувачеві.

- Рівень доступу до даних — місце, де зберігаються дані, найчастіше з використанням SQL або NoSQL рішень. Це рівень, з якого здійснюється доступ до даних та їх надсилання.

Інші моделі можуть мати менше або більше рівнів, але незалежно від того, яке архітектурне рішення було прийняте, користувачі завжди матимуть спосіб взаємодії з програмою, спосіб доставлення даних, основну систему обробки, яка займається обчисленнями, і місце, де зберігаються дані.

Для реалізації структури власного проєкту, мною було обрано Clean Architecture. Основним правилом функціонування даної архітектури, є правило залежності. Тобто між рівнями програмного продукту, встановлюється зв’язок направлений в середину. Кожен шар може взаємодіяти лише з шаром, який на рівень нижче нього, але не навпаки. За принципом взаємодії, даний шаблон проєктування, нагадує Onion та Hexagonal архітектури, на основі яких він був створений. Роберт С. Мартін представив структуру архітектури у вигляді 4 рівнів (див. Рис. 2.1).

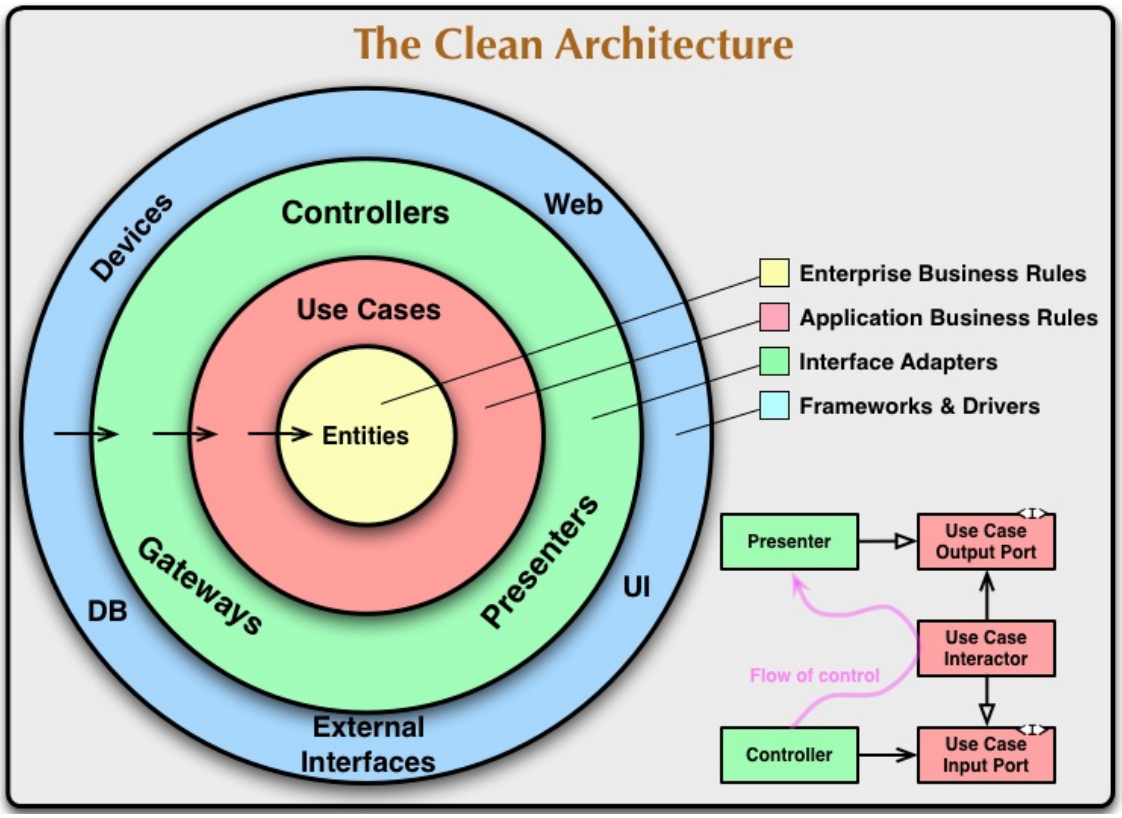

Рис. 2.1. Діаграма представлення Clean Architecture.

Джерело: [3].

Як пояснює сам Мартін, «…кола схематичні. Ви можете виявити, що вам потрібно більше, ніж лише ці чотири. Немає правила, яке говорить, що ви завжди повинні мати лише ці чотири. Однак завжди застосовується правило залежності . Залежності вихідного коду завжди спрямовані всередину. У міру просування всередину рівень абстракції зростає. Крайнє коло — бетонна деталь нижнього рівня. В міру того, як ви просуваєтеся всередину, програмне забезпечення стає більш абстрактним і інкапсулює політики вищого рівня. Внутрішнє коло є найзагальнішим.» [3].

2.2.2. Стек технологій

Також було прийнято рішення змінити стек технологій, завдяки якому буде реалізовано весь необхідний функціонал. Саме завдяки правильно підібраним

технологіям буде забезпечено якісь та надійність застосунку. Для нашої системи ми обрали наступні:

C# — сучасна, об'єктноорієнтована та безпечна мова програмування, яка працює у середовищі .NET. Вона дозволяє розробникам створювати безпечні та надійні програми із використанням різних типів. C# має багато спільного з мовами C, C++, Java та JavaScript, що робить її знайомою для програмістів, які працювали з цими мовами. C# є об'єктноорієнтованою та компонентно-орієнтованою мовою програмування. Вона надає мовні конструкції, які підтримують ці концепції, що робить C# природною для створення та використання програмних компонентів. З моменту свого створення C# продовжує розширюватись, додавати нові функції та підтримувати сучасні практики проєктування програмного забезпечення. Його сутність полягає в об'єктноорієнтованому підході, де ви визначаєте типи та їх поведінку [4].

ASP.NET Core Blazor — це технологія, яка дозволяє розробникам створювати вебдодатки за допомогою мови C# з платформою .NET Core. Для створення інтерактивних вебсторінок Blazor використовує такі вебстандарти, як HTML, CSS і JavaScript. За допомогою неї можемо створювати наступні речі, а саме:

- Створювати цікаві та інтерактивні інтерфейси, використовуючи C# замість JavaScript;
- Розділяти серверну та клієнтську логіку програм .NET;
- Розробка користувацьких інтерфейсів на HTML і CSS для різних браузерів, включаючи мобільні;
- Робити інтеграцію з сучасними платформами хостингу;
- Створювати гібридні настільні та мобільні програми за допомогою .NET і Blazor;

Blazor дозволяє створювати програми на стороні клієнта, які працюють у браузері за допомогою C# та .NET. Це дозволяє використовувати знайомі мови

програмування та інструменти для розробки вебдодатків. Blazor також має два режими роботи: серверний Blazor і WebAssembly Blazor. Blazor на стороні сервера працює на стороні сервера та використовує SignalR для забезпечення зв’язку між сервером і браузером. У цьому режимі Blazor використовує серверну модель відтворення, де компоненти Blazor відображаються на сервері та надсилаються в браузер [5].

WebAssembly Blazor працює на стороні клієнта та використовує WebAssembly для виконання коду C# безпосередньо в браузері. У цьому режимі Blazor використовує клієнтську модель відтворення, де компоненти Blazor відображаються безпосередньо в браузері. Blazor має ряд переваг, зокрема:

- Має єдиний пакет технологій, це означає що можна використовувати знайомі мови та інструменти, такі як C# і Visual Studio, для розробки вебдодатків;
- Blazor використовує WebAssembly для виконання коду C# безпосередньо в браузері, що забезпечує високу продуктивність і швидкість програми;
- Blazor дозволяє розробникам створювати безпечні вебдодатки;

Dependency injection — в .NET залежності можуть бути визначені як відношення між компонентами програми, де один компонент використовує інший. Залежності можуть бути визначені як прямі (коли один компонент прямо використовує інший) або непрямі (коли компонент використовує інший через посередника). Якщо залежність не визначена правильно, може виникнути проблема при зміні внутрішньої реалізації компонента. В .NET залежності можуть бути визначені за допомогою Dependency Injection (DI), який є шаблоном проєктування, що дозволяє відокремлювати компоненти програми один від одного і забезпечувати можливість змінювати поведінку програми без необхідності змінювати її код. DI передає залежні об'єкти як параметри конструктора або властивості, замість того, щоб створювати їх в середині компонента.

MediatR — це бібліотека, яка допомагає відокремити бізнес-логіку від коду інтерфейсу користувача. MediatR використовує патерн проєктування Mediator

Pattern, який дозволяє взаємодіяти між компонентами програми через об'єкти-медіатори. MediatR можна використовувати разом з Blazor Server, якщо весь код виконується на сервері. Для цього потрібно додати MediatR до проєкту Blazor Server, встановити пакет MediatR.Extensions.Microsoft.DependencyInjection та налаштувати контейнер DI. Після цього ви можете використовувати MediatR для надсилання запитів та повідомлень між компонентами програми.

FluentValidation — у вебдодатках ASP.NET Core можна використовувати FluentValidation для перевірки вхідних моделей, і для цього існують два основні підходи:

- Перший — це ручна перевірка, яка містить вставку валідатора в контролер або кінцеву точку API, його виклик і подальшу реакцію на результат. Це простий і надійний спосіб.

- Другий підхід — автоматична перевірка, яка підключає FluentValidation до конвеєра перевірки, що входить до складу ASP.NET Core MVC, і дозволяє перевіряти моделі до виклику дії контролера (під час прив'язки моделі). Цей підхід більш плавний, але має кілька недоліків, таких як відсутність підтримки асинхронних правил, працездатність лише з контролерами MVC і сторінками Razor, а також складність налагодження/усунення несправностей. Зазвичай автоматичну перевірку не рекомендують для нових проєктів, але вона все ще доступна для застарілих реалізацій.

AutoMapper — це проста бібліотека, яка допомагає нам перетворювати один тип об’єкта в інший. Це об’єктно-об’єктне відображення на основі конвенцій, яке потребує дуже невеликої конфігурації. Відображення об’єкта на об’єкт працює шляхом перетворення вхідного об’єкта одного типу на вихідний об’єкт іншого типу [9].

Entity Framework (EF) Core — це легка, розширювана, відкрита та мультиплатформна версія популярної технології доступу до даних Entity Framework [6]. EF Core може служити в якості об'єктно-реляційного маппера який:

- Дозволяє працювати з базою даних, використовуючи об'єкти .NET;
- Усуває необхідність у написанні більшої частини коду доступу до даних;
- EF Core підтримує багато механізмів баз даних;

В EF Core доступ до даних здійснюється за допомогою моделі. Модель складається з класів сутностей та об'єкта контексту, який представляє сеанс роботи з базою даних. Контекстний об'єкт дозволяє запитувати та зберігати дані. Для отримання додаткової інформації див. розділ Створення моделі. EF підтримує наступні підходи до розробки моделей:

- Згенерувати модель з наявної бази даних;
- Вручну закодувати модель відповідно до бази даних;
- Після створення моделі можна використати EF Migrations для створення бази даних на основі моделі. Міграції дозволяють розвивати базу даних в міру того, як змінюється модель;

IdentityModel — це бібліотека для автентифікації та авторизації в .NET, яка допомагає створювати безпечні та ефективні програми. Вона надає інтерфейси для взаємодії з різними протоколами безпеки, такими як OpenID Connect, OAuth 2.0, SAML і WS-Federation. IdentityModel надає безліч інструментів для обробки токенів автентифікації, включаючи перевірку підпису, валідацію часу життя токена та перевірку прав доступу. Бібліотека дозволяє здійснювати автентифікацію та авторизацію з використанням різних видів ідентифікаторів, таких як імена користувачів та паролі, криптографічні ключі, сертифікати та багато іншого.

Окрім того, IdentityModel може бути використана для створення власних серверів автентифікації та авторизації, що дозволяє розробникам максимально контролювати цей процес у своїх додатках. Бібліотека підтримує різні платформи, такі як .NET Framework, .NET Core, Xamarin і Universal Windows Platform.

JSON Web Tokens (JWT) - це стандарт для безпечної передачі інформації між сторонами у вигляді JSON-об'єкта. Він складається з трьох частин: заголовка, корисного навантаження та підпису. Заголовок і корисне навантаження кодуються Base64Url, а підпис — це хеш закодованого заголовка і корисного навантаження з використанням секретного ключа. У контексті ASP.NET Core автентифікація JWT — це спосіб автентифікації користувачів і захисту ваших вебаплікаторів. Коли користувач входить в систему, він отримує токен JWT, що містить інформацію про його особу та дозволи. Потім він може використовувати цей токен для доступу до захищених кінцевих точок API. Сервер перевіряє токен, щоб переконатися, що він не був підроблений і все ще дійсний. Якщо токен дійсний, сервер дозволяє користувачеві отримати доступ до захищеної кінцевої точки [7].

JWT-аутентифікація має кілька переваг над традиційними методами автентифікації, такими як файли cookie або сесійні токени. Токени JWT можна легко передавати між сторонами через Інтернет, що робить їх ідеальними для вебаплікаторів. Вони також є автономними, тобто серверу не потрібно підтримувати стан сеансу. Нарешті, вони легко декодуються і перевіряються, що дозволяє серверу швидко аутентифікувати користувачів без необхідності використання дорогих криптографічних операцій. Таким чином, автентифікація JWT — це безпечний і ефективний спосіб автентифікації користувачів і захисту вебінтерфейсів в ASP.NET Core.

Swagger (OpenAPI) — це специфікація, що не залежить від мови для опису REST API. Це дозволяє як комп’ютерам, так і людям зрозуміти можливості REST API без прямого доступу до вихідного коду [8].

2.3. Криптографічне та алгоритмічне забезпечення

Зі стрімким розвитком інформаційних технологій з'явилася потреба в алгоритмах шифрування даних, щоб гарантувати конфіденційність інформації та забезпечити захист від крадіжки та розповсюдження приватної інформації.

Криптографічні алгоритми шифрування використовуються для перетворення зрозумілого для людини тексту в шифрований, який не можливо зрозуміти без використання ключа шифрування. Шифрування забезпечує конфіденційність даних, оскільки навіть якщо зловмисник отримає доступ до зашифрованих даних, він не зможе розшифрувати їх без відповідного ключа.

Проєкт “Replica” також потребує використання криптографічних алгоритмів шифрування даних. Процес авторизації користувача в системі потребує захисту, для цього буде використано технологію JWT (Json Web Token). Це відкритий стандарт RFC 7519, який використовується для безпечної передачі інформації між сторонами у вигляді JSON об'єктів. JWT підтримує різні стандарти шифрування, такі як:

- Алгоритми ряду HMAC-SHA: наприклад, HS256 (SHA-256), HS384 (SHA-384), HS512 (SHA-512). Вони використовують спільний секретний ключ для створення та перевірки підпису.
- Алгоритми ряду RSA: наприклад, RS256, RS384, RS512. Вони використовують асиметричні ключі, приватний ключ для підпису, публічний ключ для перевірки.
- Алгоритми ряду ECDSA: Наприклад, ES256, ES384, ES512. Вони також використовують асиметричні ключі (еліптичні криві) для створення та перевірки підпису.

Процес шифрування JWT необхідний для підтвердження валідності ключа. Тобто перевірки сервером проєкту чи токен згенерований токен є валідним та згенерований саме ним. Якщо не застосувати подібних заходів безпеки, зловмисники зможуть генерувати токени на основі того, який генерується алгоритмом сервера і за допомогою згенерованого токена видавати себе за іншу людину.

Також для більш надійного захисту даних користувача, паролі в базу даних записують в зашифрованому вигляді. Для цього використовують різні методи, проте найпопулярнішими є алгоритми одностороннього хешування, так звані hash-функції.

Для хешування паролів в проєкті “Replica” доцільно використати алгоритм ряду HMAC-SHA. А саме SHA-256 який має довжину вихідного хеш-коду у 256 біт (або 32 байти). Сам процес хешування пароля містити алгоритм перетворення текстової стрічки на хеш-код та подальше перетворення хешу в шістнадцяткове представлення (див. рис. 2.3).

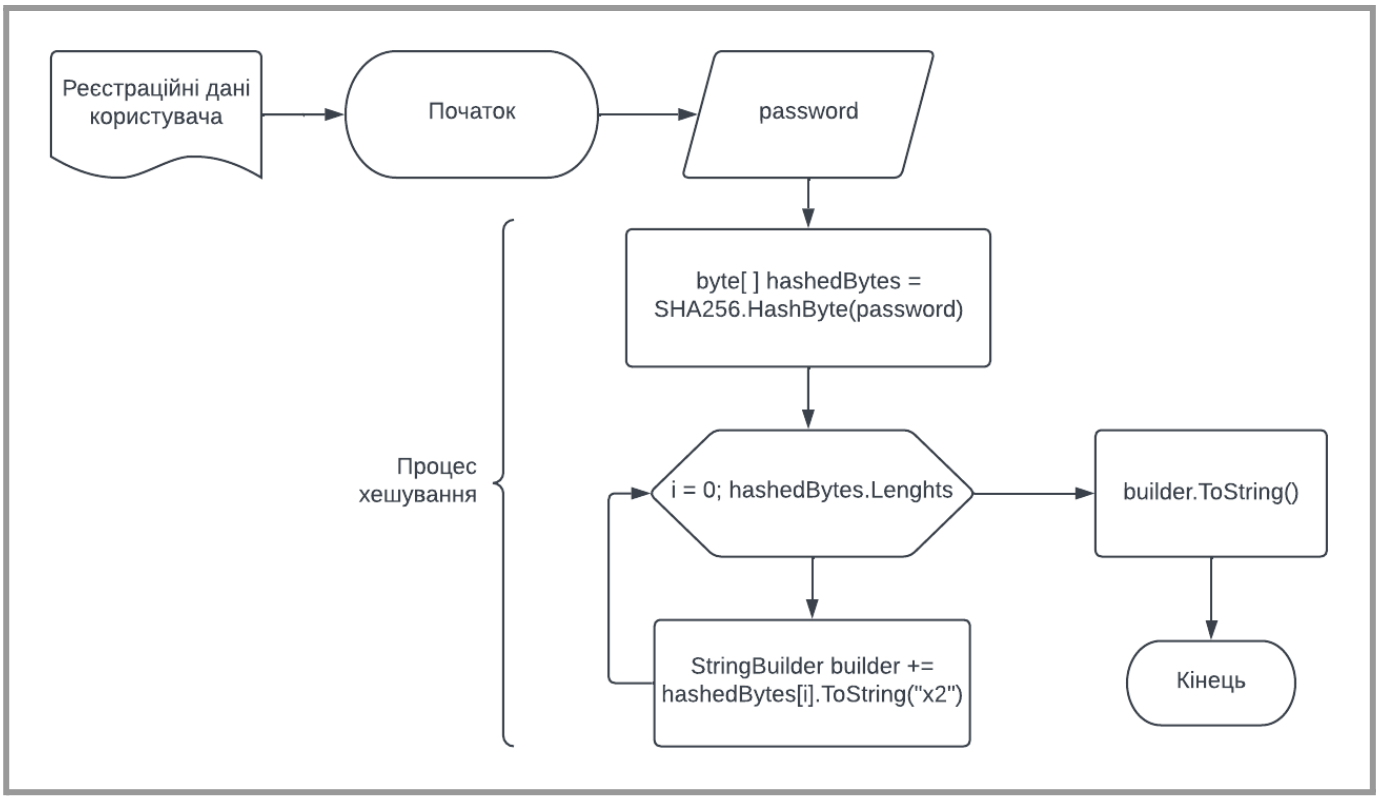

Рис. 2.3. Діаграма процесу хешування паролів алгоритмом SHA-256.

Джерело: [Створено автором].

Розберемо алгоритм хешування представлений на рисунку вище. В .NET є клас SHA256 який реалізовує алгоритм хешування, тому писати власноруч даний алгоритм недоцільно. Нам потрібно лише застосувати метод HashByte який надає клас SHA256 на вхідних даних (пароль користувача). Результатом виконання даної функції буде масив з 32 байтів, який буде збережено в байтовому масиві для подальшої обробки. Після цього кожен байт буде оброблено в циклі з використанням функції ToString("x2"). Формат "x2" вказує на шістнадцяткове представлення байта з двома символами (наприклад, 0A або FF). Шістнадцяткове представлення обов’язкове для використання, адже на виході кожен байт буде являти собою числову

змінну від 0 до 256. Після чого кожен оброблений байт буде додано в стрічку з використанням класу StringBuilder. На виході ми отримуємо текстову змінну з 64 символів (32 байти в шістнадцятковому представленні, кожен байт відповідає 2 символам).

Якщо не використовувати шістнадцяткове представлення, стрічка з бітів не матиме сталого розміру і буде складатися лише з цифрових значень. Для прикладу використаю hash-функцію до слова “EMMIT” без шістнадцяткового представлення та з ним (див. рис. 2.4).

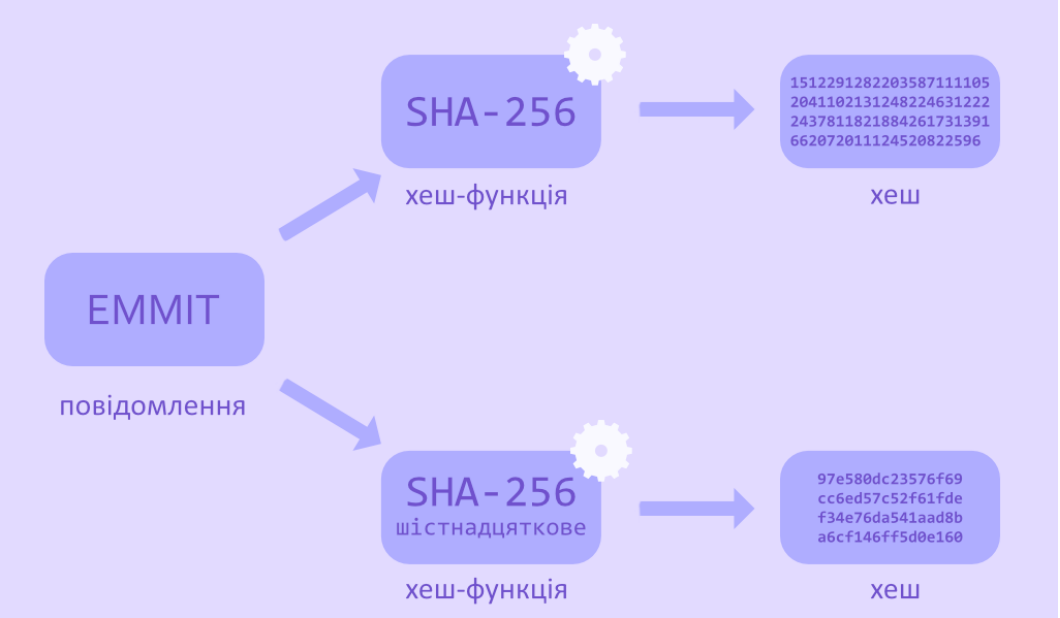

Рис. 2.4. Приклад хешування слова “EMMIT” алгоритмом SHA-256 з шістнадцятковим представленням та без нього.

Джерело: [Створено автором].

Різниця в тому, що при шістнадцятковому представленні, наш хеш матиме завжди сталу довжину в 64 символи. Це забезпечує лише краще візуальне представлення. Але при використанні будь-якого з цих двох представлень забезпечується великий простір можливих хеш-кодів і дуже мала ймовірність зіткнення (двох різних вхідних повідомлень, які мають один і той же хеш-код). Наприклад використаю функцію для хешування свого ім’я з великої та малої літери для демонстрації (див. рис. 2.5).

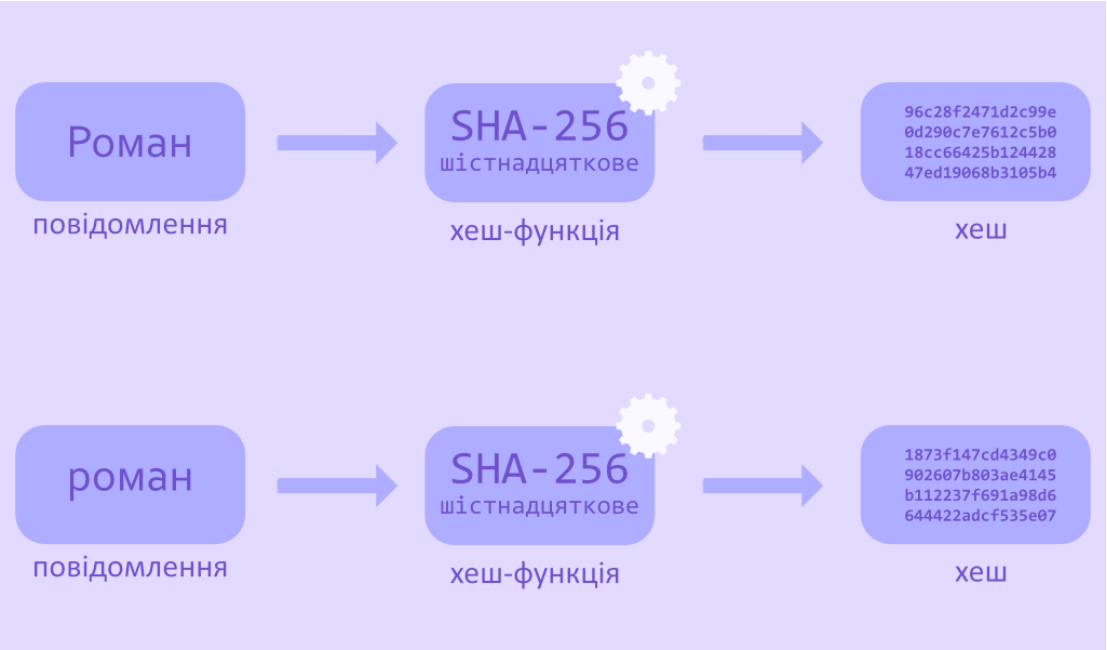

Рис. 2.5. Приклад хешування імені з першою літерою в верхньому та нижньому регістрах, алгоритмом SHA-256 з шістнадцятковим представленням.

Джерело: [Створено автором].

Як можемо бачити з рисунка вище, змінивши лише регістр першої літери на нижній отримуємо зовсім інший результат. Враховуючи це, навіть якщо зловмисник отримає хешований пароль користувача, він не зможе ним скористатися. Адже SHA-256 це алгоритм одностороннього хешування, навіть якщо зловмисник дізнається яким алгоритмом було хешовано пароль, відновити його назад не вийде.

Висновки до розділу

В процесі опрацювання даного розділу ми дослідили процеси які відбуваються в лаунж-барах під час формування та виконання замовлення. Визначили які дані необхідні для ідентифікації клієнта який здійснює замовлення. Здійснили аналіз процесу отримання від клієнта інформації про замовлення та спроєктували схему отримання такої ж інформації, проте з використанням розроблюваної нами системи. Також визначили потребу реалізації статусу обробки замовлень для зручності персоналу та інформування клієнта. Також нами був здійснений аналіз збору та обробки даних, в результаті чого ми дійшли висновків про встановлення вимог до

вхідних та вихідних даних. Завдяки впровадженню валідації даних та розподілу користувачів за рівнями допуску до інформації, система отримає захист від крадіжок та маніпуляції даними.

Здійснили проєктування системи управління замовленнями. Визначили стек технологій необхідний для створення проєкту “Replica”. Завдяки поєднанню даних програмних та технологічних рішень, буде створено надійну та стійку до відмов систему з візуально приємним та інтуїтивно зрозумілим клієнтським інтерфейсом. А також архітектурного рішення, яке відіграє важливу роль в процесі розробки програмного продукту. А також в подальших оновленнях застосунку.

Визначили потребу в шифруванні даних для забезпечення конфіденційності інформації користувачів. А також описали важливість використання hesh-функції для перетворення паролів в хеш для подальшого зберігання в системі. Завдяки цьому буде забезпечено додатковий захист від крадіжки даних.

РОЗДІЛ 3
ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1. Засоби розробки

Visual Studio (VS) є одним з найпопулярніших інтегрованих середовищ розробки (IDE) на ринку. Він розроблений компанією Microsoft і надає розширений набір інструментів для програмістів, які працюють з різними мовами програмування, платформами та технологіями.

Основні особливості Visual Studio:

1. Розширені мови програмування: Visual Studio підтримує широкий спектр мов програмування, включаючи C#, Visual Basic, C++, F#, Python, JavaScript, TypeScript і багато інших. Кожна мова має своє власне середовище засобів і підсвічування синтаксису.

2. Інструменти для розробки: VS надає розширений набір інструментів для підтримки розробки, таких як редактор коду з підсвічуванням синтаксису, автодоповненням, відступами та візуальним форматуванням. Він також має інтегровану систему керування версіями, засоби налагодження коду, аналізу продуктивності, автоматичне тестування та багато інших.

3. Розробка для різних платформ: Visual Studio дозволяє розробляти програмне забезпечення для різних платформ, включаючи Windows, macOS, Linux, iOS, Android, Azure та вебплатформи. Він надає набір інструментів і шаблонів проєктів для створення додатків під ці платформи.

4. Розширюваність: Visual Studio має потужну систему розширення, яка дозволяє розробникам створювати власні додатки, розширювати функціональність IDE, додавати нові мови програмування та інструменти. Це дозволяє використовувати Visual Studio для розробки в різних галузях індустрії.

5. Інтеграція з іншими інструментами: Visual Studio інтегрується з іншими популярними інструментами розробки, такими як Git, Docker, Azure, SQL Server, та інші. Це полегшує роботу з різними технологіями і послугами, що використовуються під час розробки програмного забезпечення.

6. Командна розробка: Visual Studio надає засоби для спільної роботи в команді розробників. Він підтримує систему контролю версій, засоби обміну кодом, інтеграцію з системами керування проєктами та іншими інструментами для спільної розробки.

Visual Studio доступний у кількох редакціях, включаючи Community (безплатна версія для некомерційної розробки), Professional і Enterprise (платні версії з розширеними функціями для комерційної розробки).

Узагальнюючи, Visual Studio є потужним інструментом для розробки програмного забезпечення, який надає багато можливостей для програмістів у різних мовах програмування і на різних платформах. Він допомагає збільшити продуктивність, спрощує розробку і спільну роботу в команді, і має широкі можливості розширення.

3.2. Вимоги до технічного та програмного забезпечення

Для створення системи управління замовленнями лаунж-бару “Replica” було впроваджено наступні вимоги до програмного забезпечення:

- Blazor WASM – фреймворк
- Visual Studio – середовище розробки
- Visual Studio Code – середовище розробки
- Adobe – графічні реактори
- C# – мова програмування

Вимоги до технічного забезпечення пристрою для розробки:

- Процесор i5 13600F
- Оперативна пам’ять 32Gb
- Відеокарта Asus RTX 3060 12Gb
- Операційна система Windows 10/11 64bit

3.3. Опис програмної реалізації

3.3.1. Серверна частина

Було прийняте рішення змінити архітектуру проєкту, оскільки попередня структура втратила вигляд чистої архітектури та набула стандартів інших архітектурних рішень. Після детальнішого ознайомлення з теорією чистої архітектури, було створено новий репозиторій з усіма необхідними на даному етапі розробки рівнями (див. рис. 3.1).

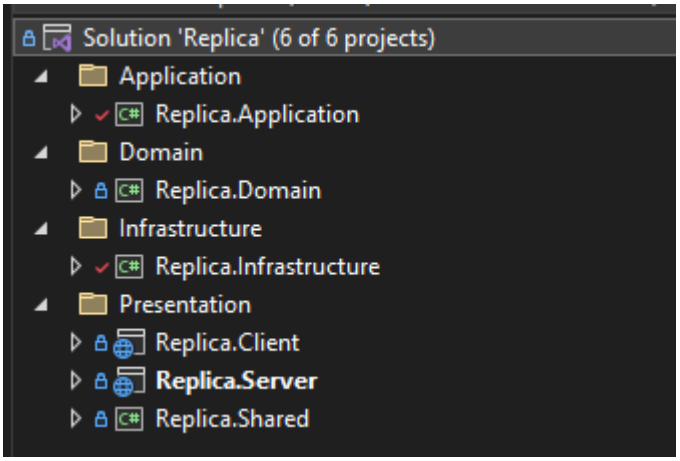

Рис. 3.1. Структура програмного застосунку для управління замовленнями лаунж-бару “Replica”.

Джерело: [Розроблено автором].

Чиста архітектура проєкту є дуже важливою з багатьох причин. Ось деякі з них:

1. Розуміння коду: Чиста архітектура дозволяє розробникам краще розуміти код, оскільки вона використовує чітко визначені шари та принципи. Це дозволяє розробникам швидше зорієнтуватися в коді та робити зміни без впливу на решту проєкту.

2. Підтримка: Чиста архітектура дозволяє проєкту бути більш стійким до змін. Оскільки кожен шар проєкту залежить лише від абстракцій, а не від реалізацій, зміни в одному шарі не впливають на решту проєкту.

3. Тестування: Чиста архітектура дозволяє краще тестувати проєкту, оскільки вона забезпечує розбиття коду на незалежні шари. Це дозволяє зручно тестувати кожен шар окремо та перевіряти правильність взаємодії між шарами.

4. Розширюваність: Чиста архітектура дозволяє легко розширювати проєкту, оскільки вона забезпечує гнучкість та модульність. Це дозволяє додавати нові функції та можливості, не впливаючи на вже наявний код.

5. Розробка: Чиста архітектура дозволяє розробникам працювати над проєктом ефективніше, оскільки вона забезпечує чітку структуру та розбиття коду на логічні шари.

Отже, чиста архітектура проєкту є дуже важливою, оскільки вона допомагає зробити код більш зрозумілим, стійким до змін, легко тестованим, гнучким та ефективним для розробки.

Було прийнято рішення змінити серверну частину програмного продукту. Розробити нову базу даних, яка зможе задовольнити вимоги до застосунку. Розмістити всі необхідні компоненти по відповідних рівнях, та більш детально ознайомитися з взаємодією рівнів чистої архітектури. Власне залишити взаємодію між базою даних та контролерами за допомогою репозиторіїв, але використати технологію MediatR та реалізувати все у вигляді команд та запитів з їхніми обробниками. Тобто додати своєрідний посередник для уникнення прямої взаємодії API контролерів та контексту бази даних абстрагувавши логіку роботи серверної частини продукту.

Також прийняли рішення про потребу в покращені та оновленні автентифікації з використанням JWT, та додаванням правильного оновлення токена доступу завдяки методиці Refresh Token. Сам Refresh Token тепер буде зберігатися відразу в

об’єкті кожного користувача, таблиці бази даних. Цей токен також не є вічним та потребує оновлень. Але поки проєкт знаходиться на стадії розробки, використання токена для оновлення JWT не буде, а час дії основного токена авторизації буде збільшено для уникнення потреби в постійному оновлені. Оскільки клієнтський інтерфейс не був реалізований і на цей момент є лише серверна частина проєкту з запитами, які необхідно виконувати в ручну.

Графічний інтерфейс знаходиться на стадії розробки та не був включений в основну роботу, тому що поки не представляє функціональних можливостей застосунку, а має лише візуальне представлення розміщення компонентів. Тобто не презентує логіку роботи додатка та авторизацію користувача.

Автентифікація користувача була реалізована за допомогою JWT, як було вказано вище. Представлення на стороні клієнта не було реалізовано, тому результат буде продемонстровано за допомогою графічного інтерфейсу Swagger. Зараз доступно два запити з контролера для автентифікації користувачів. Перший це Login для входу вже зареєстрованих користувачів до системи, другий Regestration для реєстрації нових користувачів. Запит на вхід до системи виглядає наступним чином (див. рис. 3.2).

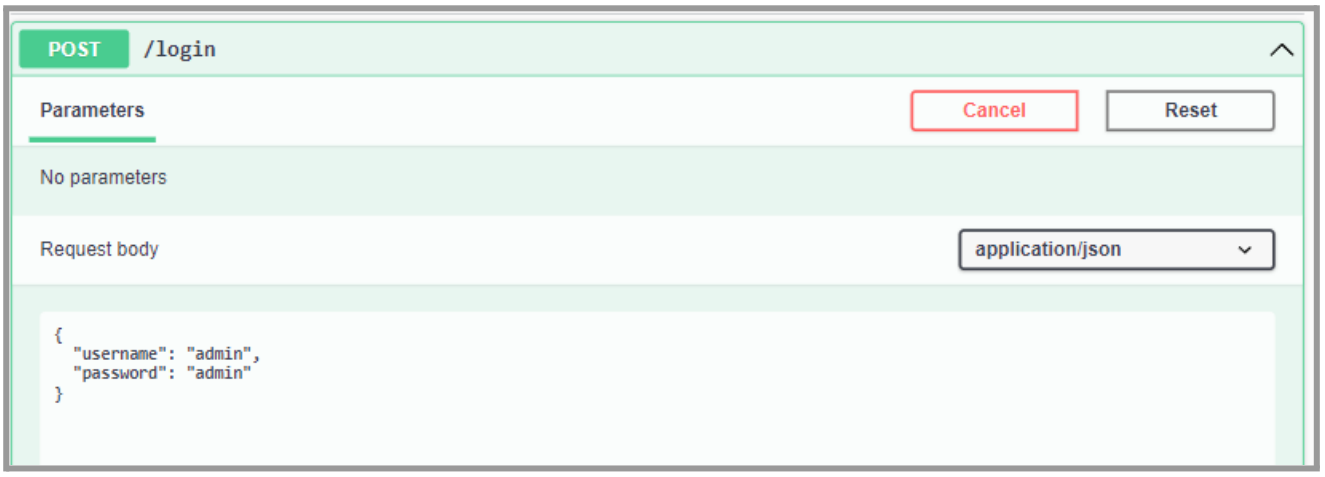

Рис. 3.2. Тіло запиту Login представлене в графічному інтерфейсі Swagger.

Джерело: [Розроблено автором].

Після надсилання запиту, вказану інформацію отримує контролер AuthenticationController (див. лістинг 3.1). Саме цей контроллер оброблятиме запити на авторизацію та реєстрацію користувачів.

Лістинг 3.1. Скорочений код класу AuthenticationController.

```
using MediatR;
using Microsoft.AspNetCore.Mvc;
using Replica.Application.Authentication.Queries.Login;
using System.ComponentModel.DataAnnotations;

namespace Replica.Server.Controllers
{
	public class AuthenticationController : ApiController
	{
	private readonly string? _apiKey;

	public AuthenticationController(IMediator mediator, IConfiguration config)
		: base(mediator) => _apiKey = config.GetValue<string>("JWT:ApiKey");

	/// <summary>
	/// Login
	/// </summary>
	/// <param name="command"></param>
	/// <response code="200">Returns user token</response>
	/// <response code="401">Invalid username or password</response>
	/// <returns></returns>
	  [ProducesResponseType(200)]
	  [ProducesResponseType(401)]
	  [ProducesResponseType(500)]
	[HttpPost("/login")]
	public async Task<ActionResult> Login([FromBody] LoginQuery command)
	{
		try
		{
		   command.ApiKey = _apiKey ?? throw new
ArgumentNullException(nameof(String));
		var response = await _mediator.Send(command);
		return Ok(response);
		}
		catch (ValidationException e)
		{
		return BadRequest(e.Message);
		}
	}
	}
}
```

Вміст класу було зменшено для представлення у звіті лише функції яку буде описано в деталях для прикладу роботи API контролерів та їх взаємодією з БД. Решта методів має подібне тіло, тому потреби в їх висвітленні немає, достатньо пояснити принципи роботи методів.

Як можемо бачити з коду наведеного вище, наш запит приймає об'єкт типу LoginQuery. Який буде містити передані користувачем для входу до системи дані. Також даний об’єкт містить секретну стрічку для генерації Verify Signature (підпису) частини JWT. Для кращого розуміння опишу структуру JSON Web Tokens.

Він складається з трьох частин:

1. Header (заголовок) — містить тип токена (typ), який зазвичай має значення “JWT”, і алгоритм шифрування (alg), який використовується для створення підпису токена.
2. Payload (тіло) — містить корисну інформацію, яка має бути передана між додатками, наприклад, ім'я користувача, його роль або термін дії токена.
3. Verify Signature (підпис) — містить зашифрований заголовок та тіло токена за допомогою обраного алгоритму шифрування та спеціального секретного слова чи набору символів (своєрідного ключа який використовується для розшифрування токена, сервер так підтверджує, що токен згенерований саме ним). Підпис забезпечує цілісність токена та перевірку на автентичність.

Усі три частини JWT розділяються крапкою (".") і кодуються в форматі Base64Url. Після кодування, JWT виглядає наступним чином: `header.payload.signature`.

Тепер можна далі розглядати роботу контролера. Створюю об’єкт інтерфейсу IMediator для роботи з командами та запитами реалізованими з його використанням. Використовуючи створений об’єкт та методу Send() який слугує для надсилання запису в відповідний обробник. Обробник запиту ми визначаємо самостійно за допомогою наслідування від інтерфейсу IRequestHandler. В ньому ж ми вказуємо

Який об'єкт він приймає та який має повернути після обробки. В нашому випадку це LoginQueryHandler (див. лістинг 3.2). Який приймає на обробку об’єкт типу LoginQuery, та повертає LoginViewModel. Обробка виконується в методі Handle, який бібліотека MediatR викликає самостійно, адже даний наслідується від інтерфейсу IRequestHandler. Забув додати, щоб все наведене вище працювало, необхідно виконати ін'єкцію залежності бібліотеки MediatR в файлі Program.cs серверу проєкту.

Лістинг 3.2. Код класу LoginQueryHandler.

```
using MediatR;
using Replica.Application.Common.Interfaces.Repositories;
using Replica.Application.Common.Interfaces.Services;
using Replica.Domain.Entities;

namespace Replica.Application.Authentication.Queries.Login
{
	public sealed class LoginQueryHandler : IRequestHandler<LoginQuery, LoginViewModel>
	{
	private readonly IUserRepository _userRepository;
	private readonly IPasswordService _passwordService;
	private readonly IJwtTokenService _jwtTokenService;

	public LoginQueryHandler(
		IUserRepository userRepository,
		IPasswordService passwordService,
		IJwtTokenService jwtTokenService) =>
		(_userRepository, _passwordService, _jwtTokenService) =
		(userRepository, passwordService, jwtTokenService);

	public async Task<LoginViewModel> Handle(LoginQuery request, CancellationToken
cancellationToken)
	{
		User user = new();

		if (request.Username is not null)
		{
		user = await _userRepository.GetUserByUsernameAsync(request.Username);
		  }

		if (_passwordService.HashPassword(request.Password) != user.Password)
		{
		throw new Exception("Invalid password.");
		}

		var token = _jwtTokenService.GenerateToken(user, request.ApiKey);
```

```
            return new LoginViewModel
            {
               JwtSecurityToken = token,
            };
      }
      }
}
```

За допомогою конструктора ініціалізую усі необхідні для роботи інтерфейси, реалізація яких знаходиться на рівні Application нашої системи управління лаунж-баром. Як бачимо з коду, обробка команди входу відбувається за допомогою простої перевірки на наявність такого користувача в системі та підтвердження вірності введеного паролю. Сам пошук користувача відбувається на рівні Application, де знаходиться клас UserRepository (написаний з використанням патерну репозиторій) для роботи з таблицею користувачів нашої бази даних, тобто власне доступ до контексту БД здійснюється саме там. Також тут присутні два сервіси, один для хешування паролів (коли користувач реєструється, пароль до бази даних запис у вигляді хешу з використанням алгоритму SHA256), другий для генерації JWT. Реалізовані ці сервіси також на рівні Application. За допомогою першого сервісу, ми перетворюємо введений користувачем пароль в хеш та порівнюємо з тим, що для нас повернувся в об’єкті користувача з бази даних. Якщо всі дані було введено вірно і такий користувач існує в системі, буде повернено об’єкт типу LoginViewModel, який містить згенерований JWT за даними користувача (див. рис. 3.3).

Рис. 3.3. Результат виконання запиту Login в графічному інтерфейсі Swagger.

Джерело: [Розроблено автором].

Можемо перевірити вміст нашого токена на сайті https://jwt.io/ це декодер для JWT. Отриманий токен додаємо в поле закодовано, декодований результат з'явиться з правої частини (див. рис. 3.4).

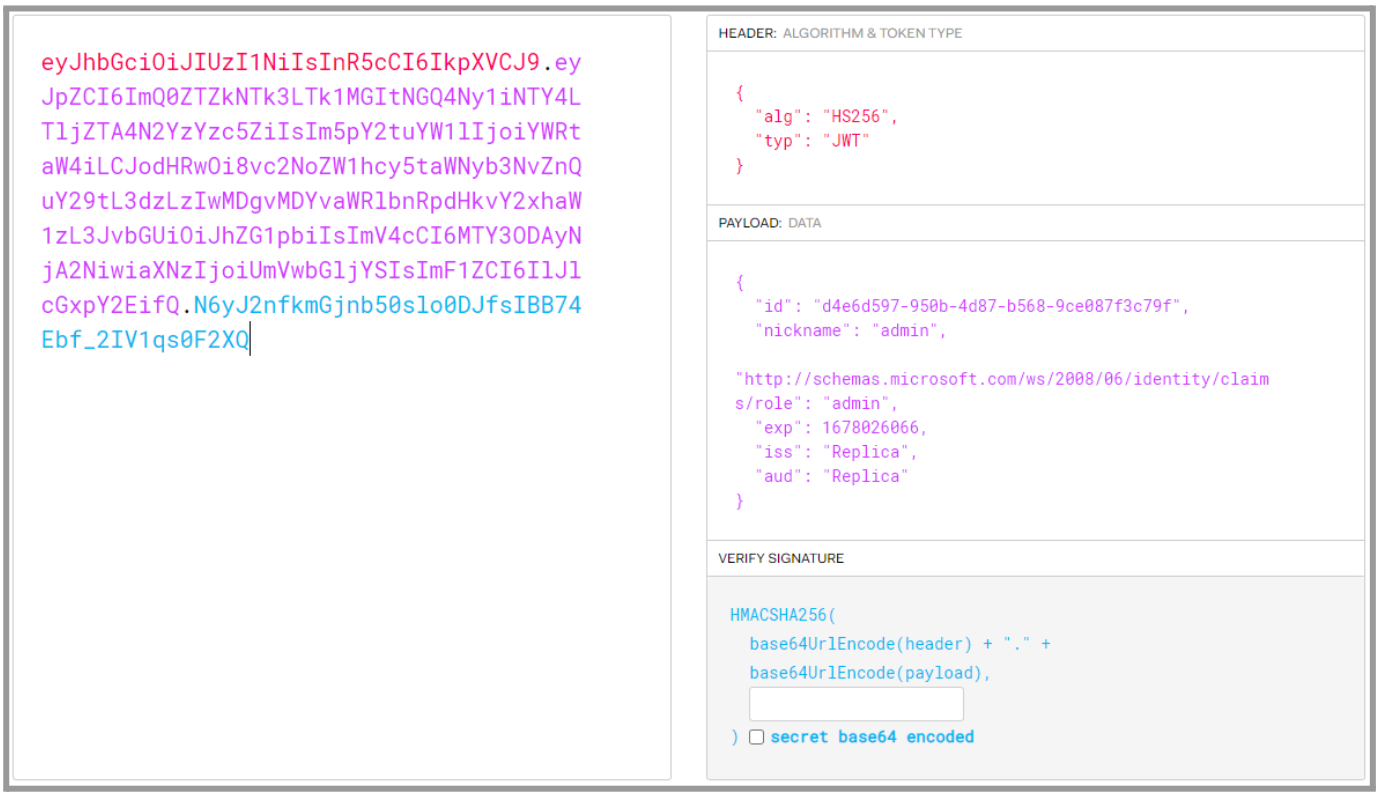

Рис. 3.4. Результат декодування токена згенерованого сервером Replica.

Джерело: [Розроблено автором].

Для виконання решти запитів потрібно використовувати токен який згенерував для нас сервер, для цього в графічному інтерфейсі Swagger (див. рис. 3.5) можна додати відповідне поле, яке дозволить вводити наш токен і виконувати запити захищені тегом Authorize якщо роль користувача дозволяє взаємодіяти з даним запитом. Тобто встановлено обмеження за роллю користувача в системі. Це зроблено для того, щоб розділити персонал та клієнтів в системі, надавши кожному конкретний перелік можливостей та обов'язків в системі. Це цілком логічна дія, адже користувач не має взаємодіяти з інтерфейсом додавання нових товарів, або підтвердження замовлень. Так можливість є лише в менеджерів та офіціантів. Своєю чергою офіціант, не має жодного відношення до інтерфейсу блокування користувачів або зміни їх ролі у системі.

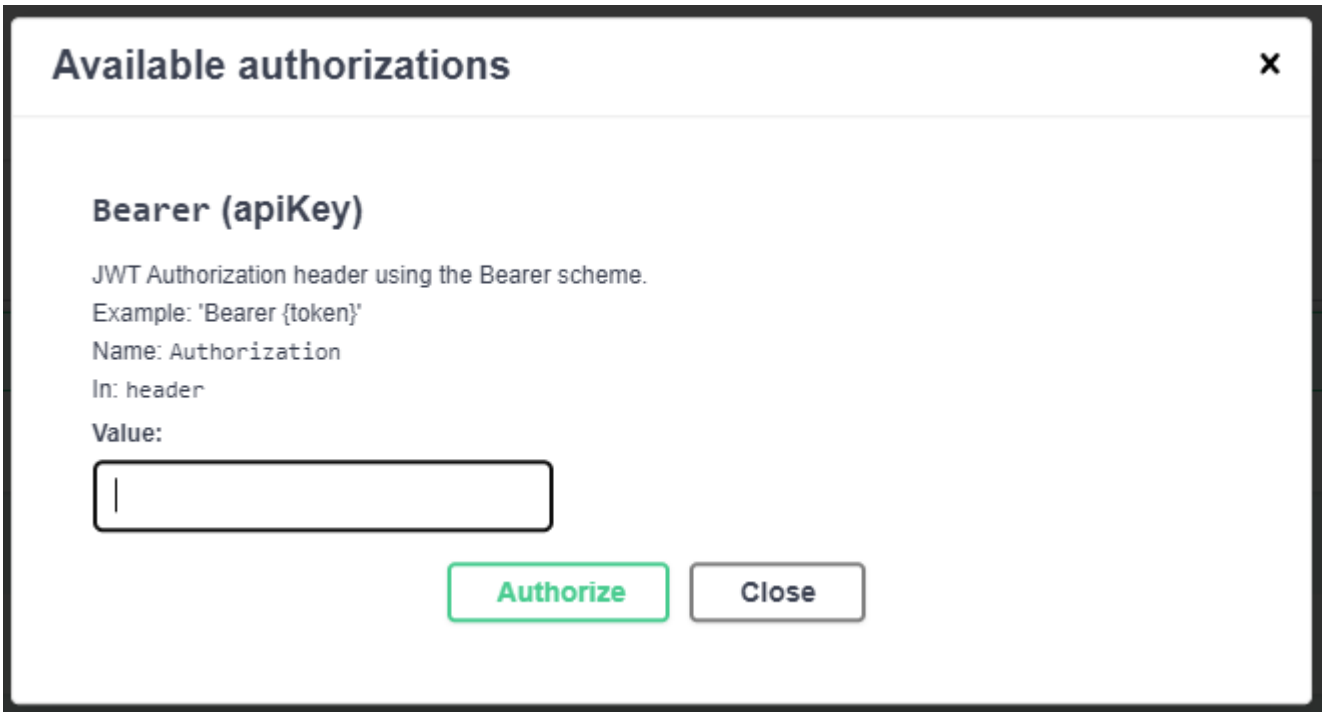

Рис. 3.5. Вікно для авторизації користувача в графічному інтерфейсі Swagger.

Джерело: [Розроблено автором].

Решту запитів до опису не додаю, оскільки логіка роботи в них подібна до наведеного прикладу. Серверна частина не описана повною мірою і продемонструвати роботу даних запитів не можливо в повноцінному вигляді. Проте

загальну логіку роботи можна описати за допомогою графічного інтерфейсу який відображає необхідні для запиту поля. На скриншоті нижче наведено приклад тіла запиту Reservation (див. рис. 3.6) для бронювання місць в закладі. Він приймає JWT токен користувача який здійснює резервування, ідентифікатор місця яке буде заброньовано, та час на котрий буде бронюватися відповідне місце.

Рис. 3.6. Тіло запиту Reservation.

Джерело: [Розроблено автором].

Тобто всі запити будуть приймати ідентифікатори товарів, користувачів та місць в залежності від типу запиту, та повертати необхідні для подальшої роботи, зрозумілі кінцевому користувачеві об’єкти.

Також як було вказано вище, для повноцінної роботи розроблюваної системи, було прийнято рішення змінити структур бази даних. Прибравши та або змінивши певні таблиці (див. рис. 3.7).

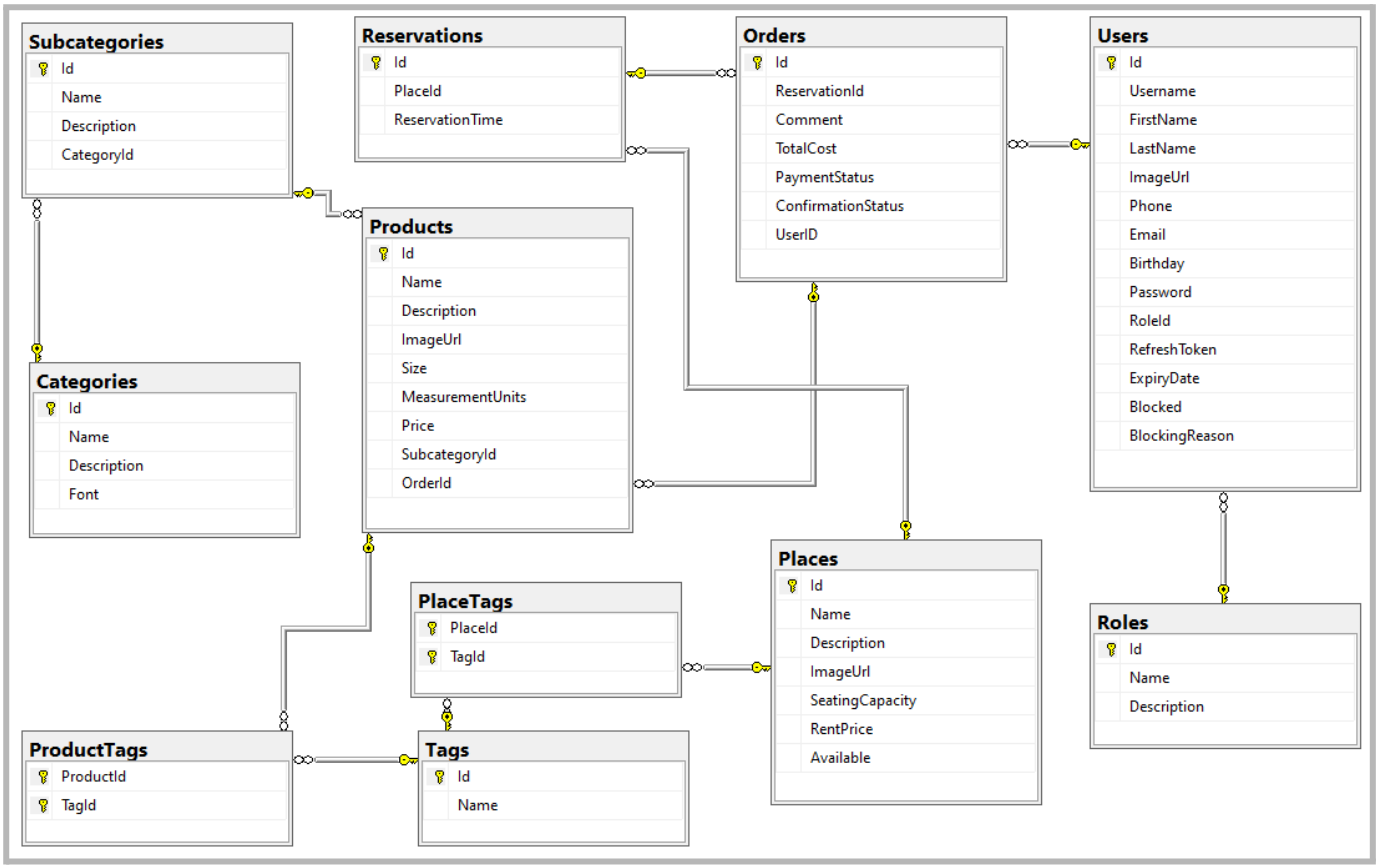

Рис. 3.7. Тіло запиту Reservation.

Джерело: [Розроблено автором].

На даному етапі розробки системи, дана БД задовольняє всі вимоги та вміщує усі неохідня для роботи сервісу дані. За потреби структура може бути доповнена або відкорегована якщо якісь дані не будуть використовуватися в фінальній версії застосунку.

3.3.2. Клієнтська частина

Для розробки проєкту "Replica" використовується фреймворк Blazor, що дозволяє розробляти клієнтську частину застосунку, використовуючи мову програмування C# і компілювати її до WebAssembly, що виконуються в браузері. Оскільки в проєкті застосовуються принципи чистої архітектури, клієнтська частина знаходиться в тому ж рішенні, що й серверна частина. Це означає, що клієнт (front-end) і сервер (back-end) розділені логічно, але фізично знаходяться в одному рішенні. Клієнтська частина зосереджена на відображенні та взаємодії з

користувачем, в той час, як серверна частина відповідає за обробку бізнес-логіки, доступ до даних і забезпечення взаємодії з базою даних або зовнішніми сервісами.

Клієнтська частина знає лише про адреси, на які потрібно надсилати запити для створення або отримання інформації. Вона не має прямого знання про внутрішню структуру серверної частини. Це дає можливість замінити фронтенд без втручання в серверну частину. Наприклад, можна замінити Blazor на інший фреймворк або навіть реалізувати мобільний додаток з використанням того ж серверного API. Цей підхід до розробки дозволяє забезпечити гнучкість і легкість внесення змін до клієнтської та серверної частини окремо, не порушуючи функціональності іншої частини.

UI-дизайн (інтерфейс користувача) важлива складова розробки фронтенду, яка відповідає за створення візуального та функціонального сприйняття користувача в процесі взаємодії з вебдодатком. Створення візуального стилю передбачає вибір кольорів, типографіки, іконок та інших елементів дизайну, які відображатимуть бренд або концепцію проєкту. Важливо забезпечити консистентність у використанні кольорів і шрифтів, щоб створити єдиний інтерфейс зі зручним сприйняттям для користувачів.

Для всього користувацького інтерфейсу було використано два шрифти: Ubuntu та Roboto. Вони мають версію для латиниці та кирилиці, тому можна вважати, що це універсальні шрифти. Вибір правильного шрифту в дизайні має велике значення і може впливати на сприйняття та ефективність комунікації з користувачем. Одним із головних факторів вибору шрифту є його читабельність, текст має легко сприйматися користувачем. Також шрифт може бути важливим елементом, який допомагає відобразити бренд або передати певну концепцію. Деякі шрифти мають властивості, що асоціюються з певними емоціями, стилем або атмосферою.

Також важливо підібрати правильні кольори для користувацького інтерфейсу. Важливо забезпечити достатню контрастність між фоном та текстом. Щоб забезпечити зручне читання, кольори мають добре відрізнятися одне від одного.

Дизайнер має підібрати правильну кольорову гаму, щоб інтерфейс виглядав приємно для користувача і відображав брендову ідентичність.

Для користувацького інтерфейсу проєкту "Replica" було обрано 4 основних кольори, які відображають брендову ідентичність і створюють відповідну атмосферу:

1. Нейтральний колір: "#FFFFFF" (Білий). Цей колір використовується для основних елементів, таких як заголовки або дрібний текст. Він створює враження чистоти, простоти та легкості.
2. Допоміжний колір: "#18181B" (Чорний мокко або Глибокий темно-сірий). Цей колір використовується для підсилення важливих елементів, або декоративних ефектів. Він додає глибини, загадковості та контрасту.
3. Акцентний колір: "#8A36E7" (Електричний фіолетовий або Насичений фіолетовий). Цей яскравий фіолетовий колір використовується для привертання уваги до особливо важливих елементів або взаємодій. Він створює враження креативності, енергії та сучасності.
4. Головний колір: "#27272A" (Чорний ебоніт або Глибокий чорний). Цей колір використовується для фону. Він створює враження стійкості, елегантності та солідності.

Використання цих кольорів в інтерфейсі "Replica" допомагає створити гармонійний дизайн, передає брендову ідентичність і відповідає сприйняттю та потребам цільової аудиторії.

Також, було розроблено логотип для системи “Replica” (див. рис. 3.8), адже він відіграє ключову роль у формуванні брендової ідентичності. Він створює візуальний символ, який в майбутньому буде асоціюватися з системою управління замовленнями лаунж-бару.

Рис. 3.8. Логотип системи управління замовленнями лаунж-бару “Replica”.

Джерело: [Розроблено автором].

Головна мета розробки клієнтської частини, це реалізація зручного та зрозумілого кожному інтерфейсу, як для клієнтів, так і для персоналу закладу. Але перше, що має зробити кожен користувач для використання системи, це пройти процес реєстрації. Тому спочатку варто розробити саме сторінку входу та реєстрації (див. рис. 3.9).

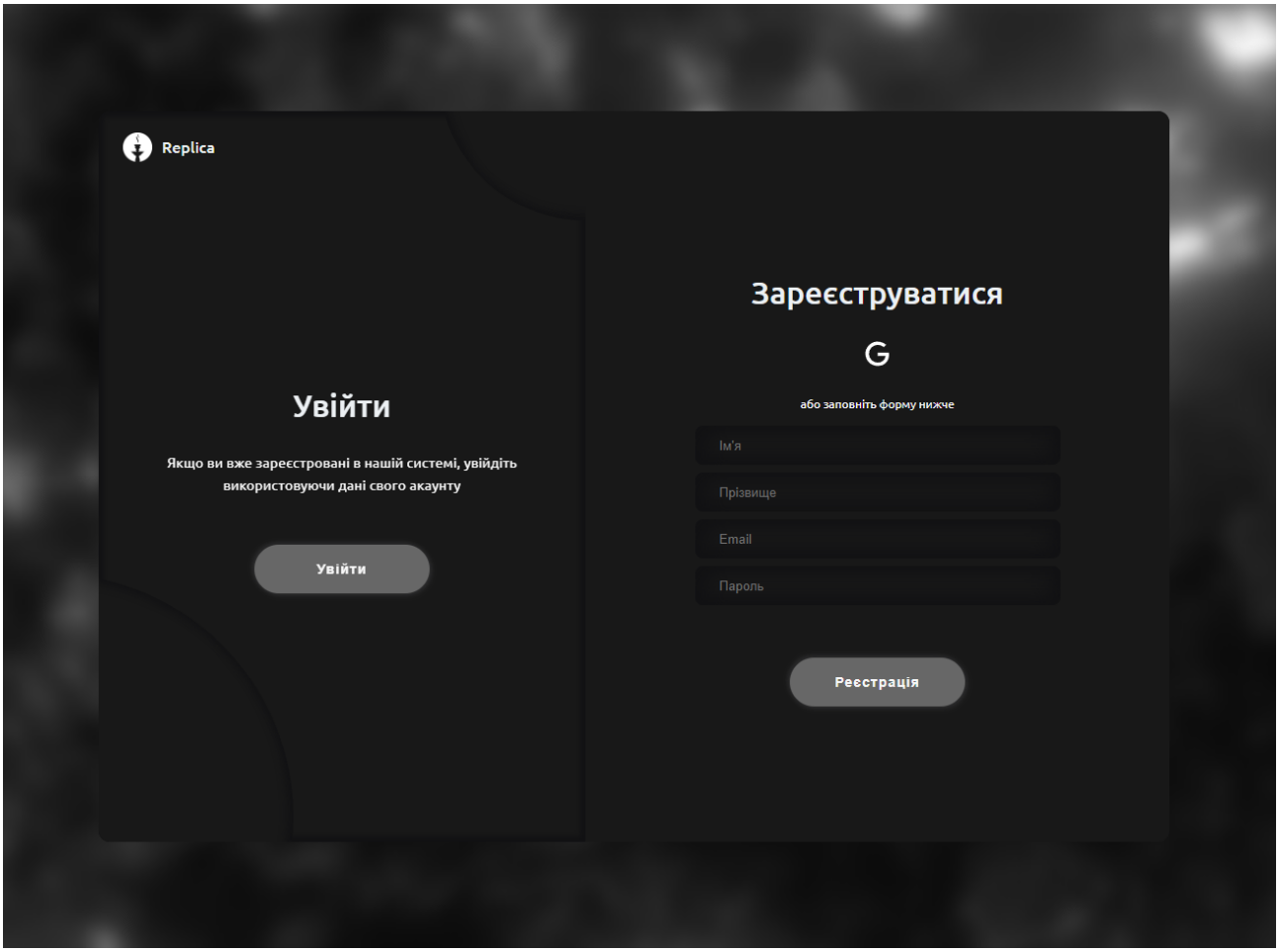

Рис. 3.9. Сторінка входу та авторизації системи “Replica”.

Джерело: [Розроблено автором].

Для використання системи потрібно більше особистої інформації ніж показано на рисунку. Проте прийнято рішення не змушувати користувача вводити велику кількість інформації під час реєстрації. Решту даних можна буде заповнити в самій системі. Головна ідея, в зменшенні кількості інформації, яку система отримує від користувача під час реєстрації це не злякати клієнта. Люди можуть бути незадоволені тим, що система на етапі реєстрації потребує значну кількість інформації, тому варто обмежитися лише певною частиною даних. Решту можна зібрати в процесі використання системи клієнтом закладу.

Авторизації з використанням Google наразі немає, проте в майбутньому дана функція буде реалізована, адже всі знайомі з перевагами такого типу авторизації в застосунках. Це економить час користувача, а значить збільшує популярність та

ефективність системи в цілому. Адже користувачеві не потрібно запам’ятовувати додаткову інформацію для авторизації, а тим паче вводити її при вході до системи. Достатньо мати акаунт google, який одночасно надасть особисту інформацію необхідну для системи.

Після реєстрації або ж входу до системи користувач потрапляє на тимчасово головну сторінку сайту, а саме “Кальян”. В майбутньому ця сторінка має бути замінена іншою, а саме сторінкою “Новини” (поки що вони є на кожній сторінці з товарами лаунж-бару), проте не було прийнято рішення остаточно про доцільність цієї сторінки. Тому поки, як головна сторінка використовується сторінка з кальянами (див. рис. 3.10) які пропонує лаунж-бар своїм клієнтам.

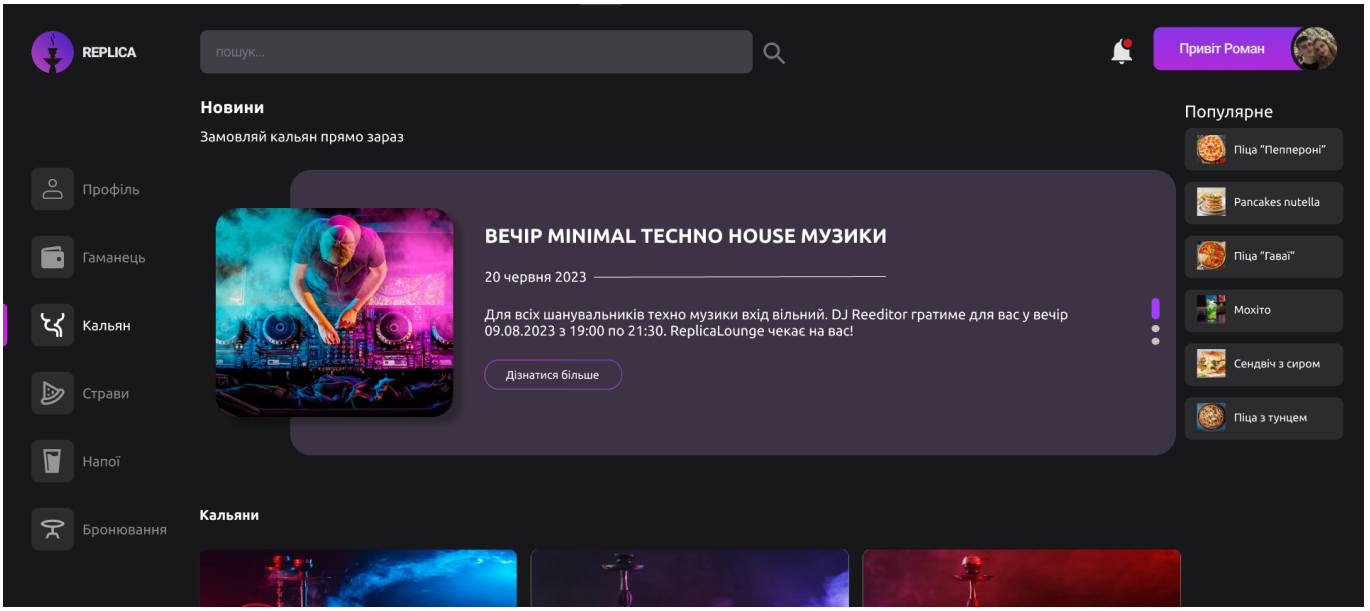

Рис. 3.10. Тимчасова головна сторінка системи “Replica”.

Джерело: [Розроблено автором].

На сторінці присутній блок з новинами, такий блок є на кожній сторінці з товарами які пропонує заклад своїм клієнтам. Проте в процесі розробки виникло питання, чи доцільно робити новини блоком який буде присутній буквально на всіх сторінках з товарами. Проте, поки рішення про потребу винесення новин на окрему сторінку не було прийнято. Головним контентом даної сторінки є список кальянів закладу відпочинку (див. рис. 3.11).

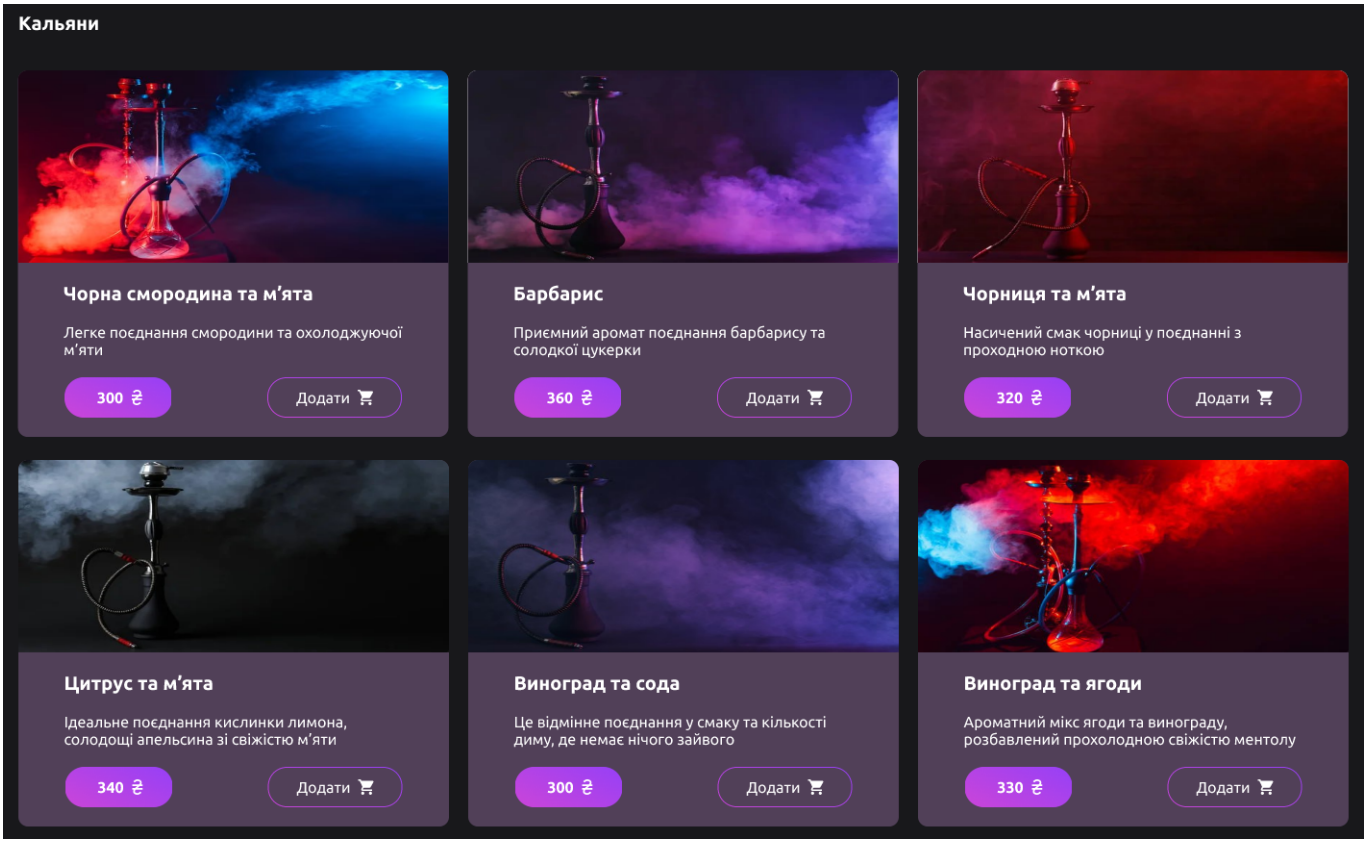

Рис. 3.11. Контент сторінки “Кальян”.

Джерело: [Розроблено автором].

Також ідентичні сторінки є для списку напоїв та страв, проте додавати графічні матеріали для ознайомлення з вмістом сторінок не є доцільним. Адже, сторінки виглядають ідентично, за винятком типу товарів. Ідентично виглядає й сторінка бронювання місць в закладі.

Дані сторінки доступні для звичайного користувача системи, тобто клієнта закладу. Для персоналу був реалізований інший інтерфейс, який відповідає їх ролі в закладі, а саме обслуговування клієнтів. Також там присутня адмінпанель яка надає змогу переглянути статистику по роботі закладу (див. рис. 3.12).

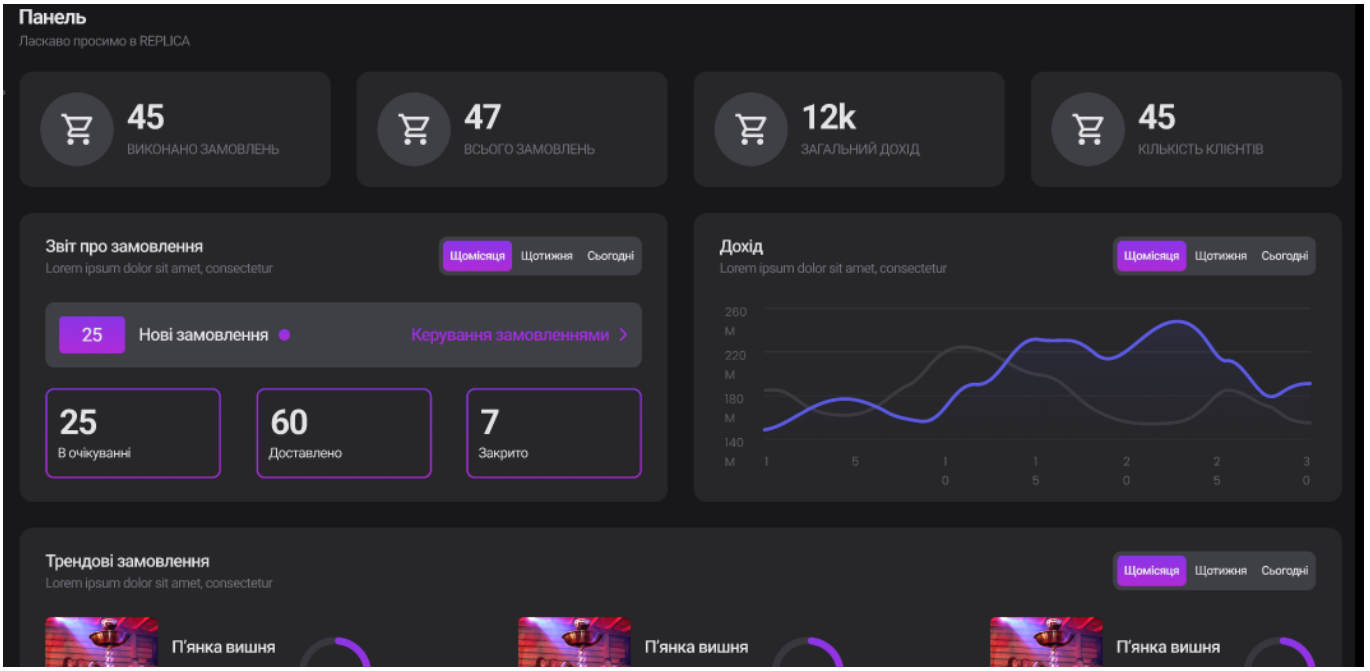

Рис. 3.12. Адмін панель системи “Replica”.

Джерело: [Розроблено автором].

На сторінці відображаються дані, про кількість клієнтів, дохід закладу, кількість замовлень. Також наявний графік для відображення доходу закладу за місяцями, тижнями та в поточний день. З низу розташований блок з популярними товарами закладу. Підтвердження замовлень відбувається на іншій сторінці (див. рис. 3.13).

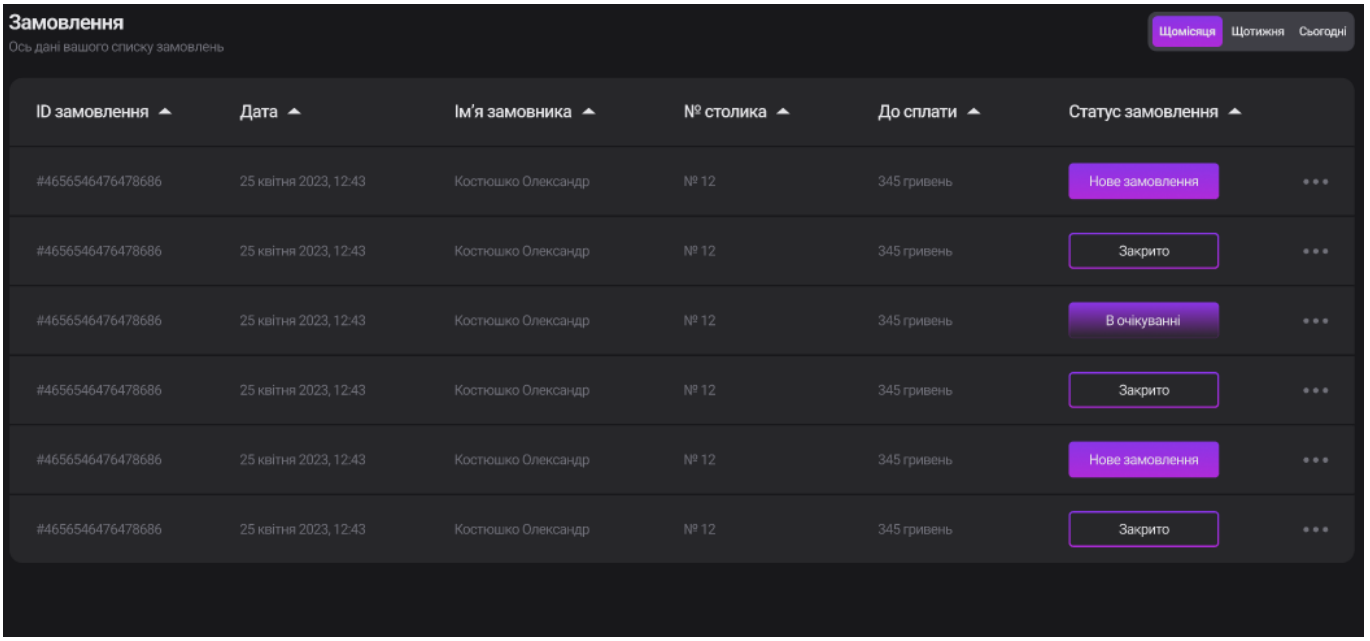

Рис. 3.13. Сторінка підтвердження замовлень системи “Replica”.

Джерело: [Розроблено автором].

Дану сторінку персонал закладу використовуватиме для підтвердження замовлень які здійснили клієнти. Замовлення відображаються у вигляді списку. Кожне замовлення містить інформацію про клієнта який його здійснив, а також статус замовлення. Замовлення розгортається і в ньому відображається список товарів які замовив клієнт, а також статус оплати.

Висновки до розділу

Ми розробили систему управління замовленнями лаунж-бару. Дана система надає змогу формувати замовлення для клієнта закладу, а персоналу здійснювати якісний контроль процесів які відбуваються в лаунж-барі. Ми реалізували зручний та інтуїтивно зрозумілий для кінцевого користувача інтерфейс. Використали новітні технології для розробки та проєктування застосунків. Для реалізації програмного проєкту “Replica” застосували підхід чистої архітектури, що забезпечить модульність для проєкту та дасть змогу за потреби замінити один з рівнів програми за потреби.

Реалізовано клієнтську частину з приємним та зручним інтерфейсом. Є дві версії клієнта які доступні в залежності від ролі користувача. Проте це досі не є кінцевою версією користувацького інтерфейсу. В майбутньому даний інтерфейс буде покращено, додано нові функції, які зроблять систему зручнішою у використанні.

В процесі розробки використали популярні IDE, такі як Visual Studio та Visual Studio Code. А також графічні редактори від компанії Adobe, завдяки яким було створено логотип проєкту “Replica”, також виконано обробку всіх необхідних для створення клієнтської частини графічних матеріалів.

ВИСНОВКИ

У даній роботі, ми висвітлили важливість розробки системи управління замовленнями лаунж-бару. Провели детальний аналіз проблематики адміністрування закладів відпочинку в цілому, використавши для цього електронні джерела інформації. Також проаналізували процеси які відбуваються в лаунж-барах для постановки конкретних завдань та виділення головних особливостей даного типу закладів відпочинку. Виділили ряд потенційних користувачів системи, та спроєктували діаграму взаємодії між персоналом закладу та клієнтом за допомогою вебзастосунку. Здійснили аналіз наявних на ринку сайтів лаунж-барів, а також систем які пропонують свої послуги готельно-ресторанному бізнесу. На основі чого, зробили висновки, що розробка системи управління лаунж-баром є унікальним рішенням та немає аналогів на ринку.

Після тривалого аналізу особливостей ведення даного роду діяльності, а саме адміністрування лаунж-бару. Було виставлено ряд вимог до програмного продукту та функцій, які мають бути реалізованими в вебзастосунку. Продовжуючи аналіз лаунж-барів, було виділено ряд сутностей, які лягли в основі проєктування бази даних. В процесі проєктування було отримано структуру, яка на даному етапі розробки застосунку, відповідає всім встановленим нами вимогам.

Для правильного функціонування та можливості тривалої підтримки вебзастосунку з варіантом майбутнього розширення функціонала проєкту, було необхідно реалізувати архітектурне рішення. Ми ознайомилися з можливими архітектурними патернами, після чого почали процес порівняння та підбору кращого з можливих варіантів, в результаті чого, було обрано Clean Architecture. Не менш важливим був підбір стека технологій на основі якого буде реалізовуватися функціонал системи управління лаунж-баром. Після чого, завершили підготовчий етап та перейшли до безпосередньої розробки програмного продукту. Спроєктована раніше схема БД, була реалізована за допомогою системи класів, які представляли сутності. Використовуючи технологію Entity Framework Core, було згенеровано БД з

усіма необхідними для повноцінної роботи вебзастосунку таблицями. Після чого, прийшли до рішення використати для реалізації рівня доступу до даних патер репозиторій, в якому описані всі необхідні для обробки даних методи. Це рішення зробило проєкт компактним. Пізніше було прийнято рішення, реалізовувати методи у вигляді команд. Проте ми прийшли до висновку, що на даному етапі в цьому немає потреби, тому лишили це як перспективу розвитку продукту.

Написання API контролерів це особливий процес, який потребує розуміння технології вебзапитів. Для цього було здійсненно ознайомлення з процесом грамотного написання та оформлення API контролерів з використанням електронних джерел. Основні функції, які ми реалізували на даному етапі, становити собою CRUD, тобто основні функції маніпуляції над даними, що дозволяють здійснити створення, редагування, видалення та повернення інформації. Також описано ряд запитів, які є важливими для виконання замовлення клієнтом. Після чого провели тестування за допомогою Swagger. Кожен запит працює не безпосередньо з сутністю, а транспортним об’єктом. Тому для швидкого перетворення одного об’єкту в інший, прийняли рішення використати технологію AutoMapper.

Одним з головних критеріїв роботи вебзастосунку, була реалізація можливості реєстрації клієнта для ідентифікації замовника тих чи інших послуг в лаунж-бару. А також можливості підтвердження замовлення за допомогою користувача з іншим, відмінним від звичайного рівнем доступу до застосунку. Для забезпечення безпеки процесу автентифікації, було прийнято рішення використати JWT. Завдяки цьому, було реалізовано формування формального ідентифікатора користувача за допомогою якого, можна підтвердити наявність того чи іншого рівня доступу до системи. Після цього, було здійснена модифікація API запитів шляхом додавання до певних запитів, атрибуту Authorize.

Система має великий потенціал і може бути адаптована до інших закладів сфери готельно-ресторанного бізнесу. Адже “Replica” це зручний інструмент для взаємодії між клієнтами та персоналом закладу, який надає можливість швидко, а головне зручно формувати замовлення та відстежувати статус їх виконання. Завдяки

особливостям системи, вона є своєрідним електронним меню з яким може ознайомитися клієнт без потреби безпосереднього перебування в закладі. А функція бронювання надає можливість клієнту резервувати місце в закладі без дзвінків до адміністрації закладу відпочинку. Власне дана система є важливою та корисною не лише клієнтам, але й працівникам закладу. Вся інформація про замовлення зберігається в системі “Replica” що дозволяє уникнути розбіжності та неточності під час замовлення. А також відстежувати оплату наданих послуг. Завдяки даному проєкту знизиться навантаження на працівників закладу і вони зможуть приділити більше уваги іншим важливим процесам, які необхідні для роботи закладу.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Common Bar Management Mistakes to Avoid. URL: https://bevspot.com/5-common-bar-management-mistakes-avoid/. (дата звернення: 15.03.2023 р.).
2. Bar Management Tips to Drive Sales and Productivity. URL: https://www.glimpsecorp.com/10-tips-for-bar-managers/. (дата звернення: 15.03.2023 р.).
3. The Clean Architecture. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html. (дата звернення: 15.03.2023 р.).
4. A tour of the C# language. URL: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/. (дата звернення: 15.03.2023 р.).
5. ASP.NET Core Blazor. URL: https://learn.microsoft.com/en-us/aspnet/core/blazor/?WT.mc_id=dotnet-35129-website&view=aspnetcore-6.0. (дата звернення: 15.03.2023 р.).
6. Entity Framework Core. URL: https://learn.microsoft.com/en-us/ef/core/. (дата звернення: 15.03.2023 р.).
7. Introduction to JSON Web Tokens. URL: https://jwt.io/introduction. (дата звернення: 15.03.2023 р.).
8. Get started with Swashbuckle and ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/tutorials/getting-started-with-swashbuckle?view=aspnetcore-6.0&tabs=visual-studio. (дата звернення: 15.03.2023 р.).
9. Getting Started with AutoMapper in ASP.NET Core. URL: https://code-maze.com/automapper-net-core/. (дата звернення: 15.03.2023 р.).

10. HTML. URL: https://developer.mozilla.org/en-US/docs/Web/HTML. (дата звернення: 15.03.2023 р.).

11. Learn to style HTML using CSS. URL: https://developer.mozilla.org/en-US/docs/Learn/CSS. (дата звернення: 15.03.2023 р.).

12. JavaScript — Dynamic client-side scripting. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript. (дата звернення: 15.03.2023 р.).

13. Create web APIs with ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-6.0. (дата звернення: 15.03.2023 р.).

14. Mattfrear. Add an authorization header to your swagger-ui with Swashbuckle. URL: https://mattfrear.com/2018/07/21/add-an-authorization-header-to-your-swagger-ui-with-swashbuckle-revisited/. (дата звернення: 15.03.2023 р.).

15. Lawand M. .Net 6 Minimal Api Authentication (JWT) with Swagger and Open API. URL: https://dev.to/moe23/net-6-minimal-api-authentication-jwt-with-swagger-and-open-api-2chh. (дата звернення: 15.03.2023 р.).

16. Asp Net Core - Rest API Authorization with JWT (Roles Vs Claims Vs Policy) - Step by Step. URL: https://dev.to/moe23/asp-net-core-rest-api-authorization-with-jwt-roles-vs-claims-vs-policy-step-by-step-5bgn. (дата звернення: 15.03.2023 р.).

17. Ardalis. Clean Architecture with ASP.NET Core. URL: https://ardalis.com/clean-architecture-asp-net-core/. (дата звернення: 15.03.2023 р.).

18. Spasojevic M. Global Error Handling in ASP.NET Core Web API. URL: https://code-maze.com/global-error-handling-aspnetcore/. (дата звернення: 15.03.2023 р.).

19. Pecanac V. 10 Things You Should Avoid in Your ASP.NET Core Controllers. URL: https://code-maze.com/ten-things-avoid-aspnetcore-controllers/. (дата звернення: 15.03.2023 р.).

20. Routing and Action Selection in ASP.NET Web API. URL: https://learn.microsoft.com/en-us/aspnet/web-api/overview/web-api-routing-and-actions/routing-and-action-selection. (дата звернення: 15.03.2023 р.).

21. Dependency injection in .NET. URL: https://learn.microsoft.com/en-us/dotnet/core/extensions/dependency-injection. (дата звернення: 15.03.2023 р.).

22. Anderson R. Introduction to Identity on ASP.NET Core. URL: https://learn.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-6.0&tabs=visual-studio. (дата звернення: 15.03.2023 р.).

23. JWT Claim Types. URL: https://identitymodel.readthedocs.io/en/latest/misc/constants.html#jwt-claim-types. (дата звернення: 15.03.2023 р.).