

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ШВИДКОЇ РОЗРОБКИ ВЕБ ЗАСТОСУНКІВ ТА
РОЗРОБКА КОНСТРУКТОРА НА ОСНОВІ LOW-CODE ПІДХОДУ
RESEARCH OF RAPID WEB APPLICATION DEVELOPMENT AND
DEVELOPMENT OF CONSTRUCTOR BASED ON A LOW-CODE
APPROACH

Виконав(ла): студент(ка) 2 курсу, групи КНТ-223м
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Системи штучного інтелекту

ГАРВОНА Д.С.

(ПРИЗВИЩЕ та ініціали)

Керівник ГЛАДКОВА О.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)
Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2024 року

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ГАРВОНА Данило Сергійович
(ПРИЗВИЩЕ, ім'я, по батькові)
1. Тема проєкту (роботи) Дослідження швидкої розробки вебзастосунків та розробка конструктора на основі low-code підходу.
Research of Rapid Web Application Development and development of a Constructor Based on a Low-code Approach
керівник проєкту (роботи) к.т.н., ГЛАДКОВА Ольга Миколаївна,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від “ 07 ” жовтня 2024 року № 398
2. Строк подання студентом проєкту (роботи) 05 грудня 2024 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Матеріали і методи. 3. Основні рішення щодо реалізації програми. 4. Експлуатація, тестування та експериментальне дослідження програми.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ГЛАДКОВА О.М., доцент		
Нормоконтролер	БЄЛОВА А.В., асистент		

7. Дата видачі завдання “ 04 ” вересня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис)

Данило ГАРВОНА

(І'мя ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Ольга ГЛАДКОВА

(І'мя ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
73 с., 3 табл., 26 рис., 3 дод., 15 джерел.

LOW-CODE, PYTHON, FASTAPI, ВЕБЗАСТОСУНОК, REST API.

Об'єкт дослідження – методи та засоби організації швидкої розробки вебзастосунків.

Предмет дослідження – програмні засоби для організації швидкої розробки вебзастосунків та конструктори на основі low-code підходу.

Мета роботи – створення low-code системи для швидкої розробки вебзастосунків.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мова програмування Python, Framework FastAPI, керування базами даних PostgreSQL.

Результати. Створено low-code систему, яка дозволяє користувачу швидко та з мінімальними затратами розробити CRM або систему обліку під потреби свого підприємства.

Висновки. Розроблено low-code систему для швидкої розробки вебзастосунків.

Галузь використання – користувачі підприємств.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 73 pages, 3 tables, 26 figures, 3 appendixes, 15 sources.

LOW-CODE, PYTHON, FASTAPI, WEB APPLICATION, REST API.

The object of the study is the methods and tools of organizing the rapid development of web applications.

Subject of the study is software tools for the organization for rapid development of web applications and the development kit based on a low-code approach.

The purpose of the work is to create a low-code system for the rapid development of web applications.

Materials, methods and technical means: object-oriented programming, Python programming language, FastAPI Framework, PostgreSQL database management.

Results. A low-code system has been created that allows the user to quickly and with minimal cost develop a CRM or accounting system for the needs of their enterprise.

Conclusions. A low-code system has been developed for rapid development of web applications.

Field of use – enterprise users.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз предметної області.....	11
1.1 Аналіз актуальності та практичності low-code підходу.....	11
1.2 Огляд існуючих low-code систем	12
1.3 Порівняння low-code платформ	13
1.4 Постанова завдання.....	15
1.4.1 Функціональність системи.....	16
1.4.2 Вимоги до інтерфейсу	16
1.4.3 Експлуатаційні вимоги	16
1.4.4 Не функціональні вимоги до продукту.....	17
2 Матеріали і методи.....	19
2.1 Вибір мови програмування	19
2.2 Вибір та обґрунтування Framework.....	22
2.3 Вибір СУБД	24
2.4 Висновки за розділом 2	27
3 Основні рішення щодо реалізації програми.....	28
3.1 Реалізація серверної частини	28
3.1.1 Автоматична генерація структури бази даних з об'єктної моделі .	29
3.1.2 Автономна публікація REST API для роботи з даними.....	31
3.1.3 Генерація схеми даних для клієнтів.....	32
3.2 Реалізація клієнтської частини	34
3.2.1 Конструювання меню веб-додатку по схемі даних.....	35
3.2.2 Відображення списків даних і форм за UI схемою	35
3.2.3 Взаємодія з серверною частиною через REST API інтерфейс	36
3.3 Висновок за розділом 3.....	37
4 Експлуатація, тестування та експериментальне дослідження програми.....	38

4.1 Призначення й умови застосування програми.....	38
4.2 Умови виконання програми	39
4.3 Розгортання застосунку	39
4.4 Алгоритм послідовності взаємодії з системою.....	39
4.5 Звернення до програми.....	42
4.6 Методика та результат тестування.....	42
Висновки	47
Перелік джерел посилання	48
Додаток А Технічне завдання	50
Додаток Б Текст програми	53
Додаток В Слайди презентації.....	66

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
CRM	– Customer relationship management;
JSON	– JavaScript Object Notation;
REST API	– Representational State Transfer Application Programming Interface;
UI	– User interface;
БД	– база даних;
ПЗ	– програмне забезпечення.

ВСТУП

Low-code — це підхід до створення, налаштування та модифікації систем і програм, який вимагає від користувача написання мінімальної кількості коду. Цей підхід можна реалізувати за допомогою платформ з низьким кодом. Вони використовують графічний інтерфейс користувача, просту логіку та інтуїтивно зрозумілі інструменти, які допомагають користувачам створювати програму [1].

Технологія low-code дає змогу компанії залишатися конкурентоспроможною, оптимізуючи виконання внутрішніх і клієнтських бізнес-процесів та підвищуючи продуктивність роботи різних відділів (не відділу розробки).

Помилково думати, що low-code/no-code – це тільки для тих, хто не вміє кодувати. Насправді розробники також використовують цю технологію, щоб позбавити себе рутини. Вони можуть виконувати менше нудних однотипних завдань, краще синхронізувати роботу розподілених команд, які працюють над одним проєктом, а також скоротити кількість помилок, спричинених людським фактором.

У сучасному світі інформаційні технології стали основою для оптимізації бізнес-процесів, автоматизації систем обліку, управління взаємовідносинами з клієнтами (CRM) і багатьох інших завдань. Однак традиційна розробка програмного забезпечення вимагає значних фінансових і людських ресурсів. Щоб подолати ці труднощі, розробники все частіше звертаються до концепції low-code підходу, який дозволяє їм створювати програми з мінімальною кількістю коду або навіть без нього [2].

Мета розробки low-code платформ — прискорити процес створення програмних продуктів, зробити їх доступними навіть для користувачів без глибоких технічних знань, а також зменшити витрати на розробку. Хоча традиційні платформи часто забезпечують графічний інтерфейс для створення програм за допомогою інструментів перетягування, моя система

концептуально відрізняється. Він не надає графічного дизайнера, але дає можливість створювати гнучкі спеціалізовані рішення, наприклад, системи CRM або систем обліку. Це дозволяє розробникам і компаніям адаптувати функціональність до індивідуальних вимог кожного проекту.

Такі платформи створені для вирішення кількох ключових завдань [3]:

- доступність для широкого кола користувачів: вони знижують бар'єр для входу в програмування та дають можливість не лише професійним програмістам, а й фахівцям з інших галузей створювати цифрові продукти;
- швидкість розробки: автоматизація завдань і можливість повторного використання шаблонів можуть значно прискорити процес розробки програмного забезпечення;
- економія ресурсів: знижуючи вартість найму розробників і скорочуючи час розробки, компанії можуть інвестувати в інші аспекти свого бізнесу;
- гнучкість налаштування: системи з низьким кодом, такі як моя, дозволяють створювати вузькоспеціалізовані програми, які можна масштабувати та оновлювати відповідно до змін у вашому бізнесі.

Такі платформи є важливим інструментом у цифровій трансформації компаній. Вони відповідають потребам сучасного ринку, де швидкість адаптації та інновації є ключовими конкурентними перевагами [3].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

У сучасному цифровому світі веб-застосунки стали найважливішим інструментом для багатьох галузей, включаючи бізнес, освіту, охорону здоров'я та комунікації. Вони забезпечують доступ до функцій та даних з будь-якого пристрою з підключенням до Інтернету, підвищуючи продуктивність та зручність користувача [4].

Процес розробки веб-додатків зазвичай вимагає значних ресурсів, включаючи час, технічні знання та фінанси. Щоб зменшити ці витрати, з'явилися інноваційні підходи до розробки, такі як Low-Code. Цей підхід надає необхідні інструменти для прискорення розвитку. У багатьох випадках програмісту надається мінімальний інтерфейс для базових операцій, а подальша настройка функціональності залежить від написання додаткового коду [5].

Таким чином, платформи такого типу стають компромісом між автоматизацією розробки та необхідністю ручної оптимізації, що дозволяє створювати рішення з урахуванням конкретних потреб. Вони корисні для компаній, які не хочуть використовувати інструменти загального призначення, такі як WordPress, і потребують більшої гнучкості [6].

1.1 Аналіз актуальності та практичності low-code підходу

У сучасних умовах швидкість та ефективність розробки веб-додатків стають критично важливими факторами успіху. Зростання цифрових послуг у бізнесі, медицині, освіті та інших галузях створює попит на інструменти, які можуть спростити розробку програмного забезпечення. Традиційна розробка потребує значних витрат часу та ресурсів та потребує кваліфікованих фахівців. Однак завдяки підходу low-code можна автоматизувати більшу частину процесу, тим самим скорочуючи час створення працюючого продукту [6].

Платформи Low-Code дозволяють швидко створювати базові прототипи або повноцінні рішення, що є важливим для компаній, які прагнуть мінімізувати витрати. У той же час, такі системи залишають місце для налаштування, коли необхідно додати унікальні функції. Наприклад, ваша система не є графічним дизайнером у повному розумінні цього слова, але надає мінімальний базовий інтерфейс. Це дозволяє користувачам розробляти спеціалізовані рішення, вручну додаючи необхідні функції [5].

Практичність таких платформ полягає в їх здатності адаптуватися до конкретних потреб клієнтів і забезпечувати баланс між простотою використання та гнучкістю. Вони особливо корисні для малих і середніх компаній, де необхідність швидкого впровадження нових продуктів часто переважає труднощі підтримки великих команд розробників. Крім того, завдяки своїй ефективності низькокодові платформи стимулюють цифрову трансформацію навіть у невеликих організаціях, які не мають великого бюджету на розвиток ІТ.

1.2 Огляд існуючих low-code систем

OutSystems – одна з найпопулярніших платформ розробки додатків на основі low-code. Вона пропонує повний набір інструментів для створення мобільних та веб-застосунків, зберігаючи при цьому доступ до повного коду для тих, хто хоче створити більш складну логіку або налаштувати функціональність під конкретні потреби. Платформа дозволяє розробникам створювати інтерфейси, інтегрувати бази даних і навіть автоматизувати процеси за допомогою візуальних інструментів, а також дозволяє використовувати компоненти користувача і писати код на JavaScript, SQL та інших мовах програмування для розширення можливостей. OutSystems націлена на компанії, яким потрібна швидка розробка додатків при збереженні високої гнучкості та масштабованості. Інтерфейс з функцією перетягування дозволяє навіть користувачам з обмеженими знаннями в

області програмування створювати інтерактивні веб та мобільні програми. Однак для більших і складніших систем, таких як інтеграція зі сторонніми системами або гнучке настроювання логіки програми, OutSystems дозволяє розробникам писати код [7].

Mendix – ще одна популярна платформа для розробки додатків на основі low-code. Це дозволяє створювати як мобільні, так і веб-додатки, не вимагаючи глибоких знань програмування. Mendix пропонує різноманітні візуальні інструменти для проектування, створення процесів, інтеграції та тестування програм. У той же час Mendix, як і OutSystems, забезпечує розширене налаштування програми та надає доступ до коду через Java, JavaScript та інші мови. Mendix призначений для середніх і великих компаній і забезпечує швидке створення прототипів і масштабування до реальних продуктів. Його можливості дозволяють використовувати низькорівневий код для загальних потреб і додавати власні рішення для конкретної бізнес-логіки або інтеграції в існуючі корпоративні системи. Крім того, Mendix підтримує автоматизоване тестування та розгортання, що робить платформу ще зручнішою для професійних розробників [8].

1.3 Порівняння low-code платформ

OutSystems і Mendix – це платформи low-code, які дозволяють користувачам створювати програми з мінімальними технічними навичками, зокрема за допомогою використання візуальних інтерфейсів і шаблонів. Вони зосереджені на швидкості розробки додатків і надають потужні інструменти для інтеграції з іншими системами, масштабування та розробки функцій для задоволення потреб бізнесу. Обидві платформи підтримують розробку програм для мобільних і веб-платформ, а також дозволяють додавати спеціальний код для більш складних завдань [7].

OutSystems дозволяє користувачам швидко створювати програми за допомогою компонентів і шаблонів, які можна перетягнути, але все ще дозволяє розробникам додавати код, якщо їм потрібно додати певну функціональність [7]. Mendix також пропонує потужні функції візуальної розробки, але також дозволяє інтегрувати власний код, що робить платформу дуже гнучкою для різних типів бізнесу з особливими потребами [8].

Моя система, на відміну від цих платформ, не надає графічного дизайнера для створення програм. Це означає, що весь процес створення програм складається з написання коду, хоча система надає деякі інструменти та шаблони, щоб полегшити цю роботу. Зокрема, це дозволяє створювати користувацькі CRM або системи обліку, більш конкретно адаптовані до потреб кожної організації. Цей підхід забезпечує набагато більшу гнучкість у пристосуванні функціональності до конкретних потреб, але вимагає від користувачів певного рівня технічних знань і досвіду програмування. Замість того, щоб покладатися на візуальні інтерфейси, такі як OutSystems або Mendix, де багато функцій можна створити за допомогою перетягування, моя система пропонує більше контролю над процесом розробки, але за рахунок необхідності писати код [7].

Незважаючи на те, що OutSystems і Mendix є дуже популярними платформами для швидкої розробки додатків у корпоративному середовищі, особливо завдяки простоті їх використання для нетехнічних користувачів, адаптивність їхнього підходу обмежена порівняно з системами, орієнтованими на код. Вони підходять для більш загальних випадків використання, коли вам потрібно швидко запустити програми з базовими функціями, але можуть бути менш гнучкими, якщо вам потрібно створити абсолютно унікальне або складне рішення, яке потребує налаштування [8].

Однак у моїй системі відсутність графічного дизайнера дозволяє користувачам створювати більш персоналізовані та складні рішення. Це робить його більш придатним для компаній, які мають особливі вимоги до функціональності своїх програм, наприклад: В. створення CRM систем або

бухгалтерських програм. Тут розробники можуть більш гнучко налаштовувати функціональність без обмежень графічних дизайнерів, але в той же час впровадження необхідних рішень вимагає більше часу та знань.

Отже, основна відмінність між такими платформами, як OutSystems і Mendix, і моєю системою полягає в підході до створення додатків: перші дозволяють швидко розробляти з мінімальними знаннями програмування, тоді як моя система дозволяє більше персоналізувати за допомогою програмування, але вимагає від користувачів технічних навичок, щоб налаштувати рішення для конкретні потреби бізнесу. Таблиця порівняння low-code платформ наведена нижче (табл. 1.1).

Таблиця 1.1 – Порівняння low-code платформ

Критерій	OutSystems	Mendix	Розроблювальний продукт
Безкоштовний	+	+	+
	(обмежений ресурс)	(обмежений ресурс)	
Графічний інтерфейс	+	+	-
Вимоги до системи	+	+	+
Швидкість навчання роботи	+	+	+
Персоналізована розробка під конкретні потреби бізнесу(CRM, системи обліку)	-	-	+

1.4 Постановка завдання

Головним завданням роботи є створення системи, яка дозволяє створювати користувацькі CRM або системи обліку, більш конкретно

адаптовані до потреб кожної організації, витрачаючи менше часу на саму розробку.

1.4.1 Функціональність системи

Функціональні вимоги серверної частини:

- генерування, по об'єктній моделі даних, структури бази даних;
- автоматична публікація REST API інтерфейс для роботи з даними;
- будування схеми даних для клієнта.

Функціональні вимоги клієнтської частини:

- конструює меню веб-додатку по схемі даних;
- за UI схемою відображає списки даних і форми даних;
- взаємодіє з серверною частиною через REST API інтерфейс.

1.4.2 Вимоги до інтерфейсу

Вимоги до інтерфейсу системи:

- інтерфейс повинен бути в приємних кольорах;
- система повинна мати зручний інтерфейс;
- інтуїтивно зрозумілий для користувача;
- продуктивно використовувати простір екрану.

1.4.3 Експлуатаційні вимоги

Для експлуатації клієнтської частини необхідні такі програмно-технічні засоби:

- процесор із тактовою частотою від 2 GHz;
- комп'ютер з графічною ОС;
- оперативної пам'яті від 4GB;

- підключення до Internet мережі.

1.4.4 Не функціональні вимоги до продукту

Не функціональні вимоги до продукту наступні:

- продуктивний хостинг;
- вебсервер Gunicorn;
- підтримка Python та FastAPI.

Для експлуатації серверної частини необхідні такі програмно-технічні засоби:

- процесор із тактовою частотою від 2 GHz;
- комп'ютер з графічною ОС;
- оперативної пам'яті від 4GB;
- підключення до Internet мережі.

1.5 Висновок за розділом 1

З метою розробки low-code платформи, в першому розділі основну увагу було приділено дослідженню проблемної області low-code систем. Спочатку було розглянуто сучасний стан проблемної області. Після цього було досліджено можливості часткової автоматизації процесу, через використання програмних засобів. На основі попереднього дослідження було виділено основних лідерів галузі та детально проаналізовано їх переваги та недоліки.

Завдяки таким платформам користувачі можуть зосередитися на вирішенні бізнес-завдань, а не на технічних аспектах програмування. Однак, порівняно з іншими платформами, така система, як представлена в цьому проекті, характеризується тим, що її можна налаштувати за допомогою додаткового кодування. Це робить його ідеальним для компаній, яким

потрібні унікальні рішення, такі як CRM або системи бухгалтерського обліку, що забезпечують баланс між автоматизацією та гнучкістю.

Підхід з низьким кодом стає не тільки трендом, але й необхідністю в сучасному бізнес-середовищі, де швидкість і адаптивність є важливими факторами успіху. Розробка таких систем вимагає інтеграції інноваційних підходів і розуміння потреб користувача для забезпечення комфорту та ефективності.

2 МАТЕРІАЛИ І МЕТОДИ

В першому розділі основну увагу було приділено дослідженню сучасний стан проблемної області.

В другому розділі буде проводитися вибір та затвердження технічних засобів розробки: технологія реалізації застосунку, мова програмування з framework, система БД MySQL . Порівняння та аналіз переваг та недоліків дозволить зробити ґрунтовний вибір.

2.1 Вибір мови програмування

Це високорівнева мова програмування, яка по праву належить до одних із найпростіших і найзрозуміліших, завдяки своєму синтаксису та читабельності. Вважається ідеальним для початківців, тому що програмувати на Python набагато швидше і простіше, ніж, наприклад, Java Script. Можна вивчити синтаксис Python, використовуючи звичайний підручник на офіційному сайті. Все досить добре написано [9].

Python – мова інтерпретується і перед запуском є просто текстовим файлом. У ньому немає етапу компіляції. Він логічний, добре спроектований та підходить для роботи практично на всіх платформах.

Крім того, Python є гарним для більшості проектів, таких як [9]:

- веб-розробка;
- застосунки;
- програмне забезпечення для мобільних пристроїв;
- вбудовані системи і багато іншого.

Якщо аналізувати те, де найчастіше використовується ця мова, то лідером, безперечно, є веб-розробка. Що стосується мобільних програм, то серверна частина зазвичай пишеться на Python, а для пристроїв Android та IOS використовуються інші мови.

Python часто є основою для відтворення вбудованих систем, таких як Raspberry Pi.

Мова використовується для створення засобів регулювання різних показників, таких як температура, тиск, витрати різних ресурсів. Тож Python широко використовується розробниками популярних систем розумного будинку.

Знання бібліотек Python дуже важливе. NiceGUI, JSON, знання алгоритмів та індивідуальні навички відкриють двері багатьох компаній. Звичайно, компаній на ринку багато, всі різні вимоги теж можуть відрізнятися, але ця мінімальна база дасть покупцеві конкурентну перевагу перед іншими [9].

Сьогодні, коли обчислювальна потужність персональних комп'ютерів і серверів стала досить високою, існує величезний попит інтерпретовані мови програмування. Адже, крім запуску самої програми, необхідно запустити інтерпретатор, що потребує додаткових ресурсів. Python дозволяє це зробити [10].

Популярність Python обумовлена тим, що ця мова здатна вирішувати широке коло завдань і її застосування можливе на всіх популярних сьогодні платформах, за винятком мобільного сегменту, де міцно вкоренилися специфічні мови. В останні роки Python набув популярності завдяки своїй ефективності в таких галузях розробки, як машинне навчання та наука про дані за рахунок своєї розширюваності та гнучкості, настільки необхідних у цьому сегменті програмування. При цьому Python широко використовується в WEB, в якому ця мова чудово почувається з початку 2000-х і продовжує розвивати і вдосконалювати цей напрям до сьогодні.

Далі докладніше буде розглянуто всі переваги та недоліки цієї мови [10].

Переваги:

- гнучкість, є головною перевагою мови, оскільки завдяки своїй гнучкості мова завоювала популярність серед багатьох розробників;

- можливість розширення — є бібліотеки та фреймворки під будь-які завдання та потреби, також величезний плюс у тому, що можна використовувати C код з Python;
- простота синтаксису — із синтаксису прибрано все зайве, код чистий і зрозумілий без зайвих дужок та виразів;
- інтерпретатор Python існує для всіх популярних платформ і включений за замовчуванням у більшість дистрибутивів Linux, а це означає, що він доступний на більшості серверів «з коробки»;
- PEP — єдиний стандарт написання коду, який робить код зручним та читабельним, навіть якщо він передається від одного програміста до іншого. Це робить Python популярним;
- Open Source — код інтерпретатора Python відкритий, що дозволяє всім, хто цікавиться розвитком мови, брати участь у її розробці та покращувати її, якщо подивитись подробиці релізу однієї з версій мови, то можна побачити, що величезні частини нового функціоналу реалізовані сторонніми розробниками;
- ком'юніті — навколо Python сформувалася досить дружна і приємна спільнота, яка готова прийти на допомогу будь-якому початківцю або вже досвідченому розробнику і розібратися в його проблемі.

Недоліки:

- продуктивність — Python не такий швидкий, порівняно, це пов'язано з тим, що Python інтерпретується, але навіть у порівнянні з іншими інтерпретованими мовами помітно, що Python програє у продуктивності, але це легко нівелюється за допомогою реалізацій C тієї чи іншої проблемної частини коду, в умовах сьогоденних потужностей не дуже помітно;
- динамічна типізація — через динамічну типізацію Python споживає більше ресурсів, ніж міг би, але це часто компенсується внутрішнім кешуванням;
- Global Interpreter Lock, в даний час це серйозна проблема з продуктивністю в Python, і вона також викликана поганою реалізацією

багатопоточності, код GIL не змінився з моменту виходу першої версії мови, що ясно вказує на те, що він застарів.

Порівняння мов програмування надане у таблиці 2.1.

Таблиця 2.1 – Порівняння мов програмування

Критерій	PHP	Python	ASP.NET
Надбудови для структуризації розробки	Багато	Багато	Мало
Серверна підтримка	Розповсюджена	Розповсюджена	Недостатньо розповсюджена
Масштабованість розробок	Проста	Проста	Проста
Складність підтримки	Низька	Середня	Середня

2.2 Вибір та обґрунтування Framework

FastAPI – це сучасний, швидкий (високопродуктивний), вебфреймворк для створення API за допомогою Python, в основі якого лежить стандартна анотація типів Python [11].

FastAPI — це структура для створення компактних і відносно швидких HTTP API-серверів із вбудованою валідацією, серіалізацією та асинхронністю, так би мовити, з коробки. Він заснований на двох інших фреймворках: Starlette обробляє мережу в FastAPI, а Pydantic відповідає за перевірку.

Основні функції FastAPI включають [11]:

- підтримка Async/await: FastAPI без проблем підтримує асинхронне програмування за допомогою синтаксису async/await Python, що дозволяє розробникам писати ефективний і масштабований код;

- підказки щодо типів: використовуючи підказки щодо типів, FastAPI забезпечує надійне введення та автоматичне документування API, зменшуючи кількість помилок і покращуючи зручність обслуговування;
- Pydantic: FastAPI використовує Pydantic для обробки даних, перевірки та (де)серіалізації, Pydantic сам по собі дуже продуктивний, і його ядро нещодавно було переписано на Rust для додаткової швидкості;
- сумісність з ASGI: FastAPI розроблений для роботи з ASGI-сумісними серверами, такими як Uvicorn, що забезпечує бездоганну інтеграцію з іншими фреймворками та бібліотеками;
- підтримка OpenAPI: FastAPI створює автоматичну документацію OpenAPI для API, що полегшує інтеграцію з інструментами та службами сторонніх розробників.

Кожен API (внутрішній чи зовнішній) потребує документації. Фактичним стандартом для документування API стає специфікація OpenAPI. Документацію в цьому форматі можна читати машиною, оскільки вона має формат JSON або YAML і може використовуватися для створення коду (для клієнтів і серверів, залежно від того, звідки походить документація). Крім того, завдяки Swagger і Redoc, його можна відобразити в зручній для читання формі як веб-сайт документації [12].

FastAPI підтримує автоматичне створення документації на основі моделей даних, які розробник визначає для запитів і відповідей, метаданих вашої кінцевої точки та деякої додаткової розмітки, яку можна поступово додавати за потреби для додаткових описів, прикладів тощо. Мінімум, який отримує розробник, це автоматичний. На додаток до стандартної документації кінцевої точки, FastAPI надає засоби документування зворотних викликів і веб-хуків, які може надавати API. Це те, що ще належить знайти в інших фреймворках [12].

Якщо все, що потрібно, це рендеринг JSON у програмі API, це (у простій формі) досягається шляхом повернення Dicts від обробників.

FastAPI має більше можливостей завдяки можливості використовувати моделі даних Pydantic як серіалізатори. Коли створюється дамп даних, то можна налаштовувати назви полів за допомогою псевдонімів, пропускати невстановлені поля та робити багато іншого [13].

Порівняння Framework наводиться у таблиці 2.2.

Таблиця 2.2 – Порівняння Framework

Критерії	FastAPI	Flask
Мова	Python	Python
HTTP methods	@app.get("/")	@app.route("/", methods=["GET"])
Відображення помилки	Помилка відображається в JSON форматі	Помилка відображається на html сторінці
Перевірка даних	Вбудована функція перевірки даних	-
Підтримка асинхронних завдань	+	-
Продуктивність	Реалізує специфікації ASGI	Обмеження в програмах WSGI
Підтримка	Менша підтримка	Більша підтримка

2.3 Вибір СУБД

PostgreSQL – це безкоштовна система керування базами даних (СУБД) із відкритим кодом. Це одна з найпопулярніших систем СУБД, і тому її використовують тисячі організацій по всьому світу [14].

PostgreSQL є проектом з відкритим кодом. Він приваблює багатьох розробників і компаній, тому що дозволяє заощадити на ліцензіях і налаштувати СУБД під свої конкретні потреби. Велика спільнота

розробників постійно працює над СУБД, щоб покращити PostgreSQL, додати нові функції та виправити помилки. Існує багато ресурсів, форумів і документації, які допоможуть вирішити проблеми.

PostgreSQL повністю сумісний зі стандартом SQL і підтримує багато розширень. Це робить його ідеальним для інтеграції з іншими системами та додатками. Крім того, користувачі можуть створювати власні функції, оператори та типи даних, що робить його надзвичайно гнучким.

PostgreSQL базується на мові SQL. PostgreSQL належить до іншого типу СУБД – об’єктно-реляційної (тобто поєднує в собі переваги реляційних баз даних і об’єктно-орієнтованого програмування).

PostgreSQL підтримує багато функцій, які роблять його потужною та гнучкою СУБД. Ось деякі з його основних функцій [14]:

- PostgreSQL забезпечує транзакції з властивостями ACID, що гарантує цілісність даних;
- підтримує автоматичне оновлення представлень, що дозволяє користувачам переглядати дані в базі даних у реальному часі;
- підтримує матеріалізовані представлення, які зберігають копію даних представлення в базі даних;
- підтримує тригери, які дозволяють виконувати дії при зміні даних у таблиці;
- підтримує зовнішні ключі для підключення таблиць;
- підтримує збережені процедури для виконання складних обчислень на стороні сервера.

База даних PostgreSQL успішно використовується в різних галузях. У веб-розробці він служить основою для динамічних веб-сайтів і програм. При аналізі даних це забезпечує високу ефективність обробки великих обсягів інформації. Масштабні програми, такі як соціальні мережі, покладаються на масштабованість і надійність цієї системи.

Наприклад, під час створення веб-сайтів (наприклад, програми електронної комерції) PostgreSQL може ефективно обробляти мільйони

транзакцій для обліку замовлень, зберігання профілів клієнтів і керування запасами, забезпечуючи стабільність і масштабованість. В аналізі даних (аналітична програма для маркетингових даних) – може зберігати й обробляти дані клієнтів, ефективно виконувати складні запити для аналізу купівельної поведінки та забезпечувати міцну основу для стратегічних рішень [14].

А у великомасштабних додатках (соціальна мережа) він може служити базою, на якій мільйони користувачів можуть взаємодіяти, ділитися вмістом і отримувати персоналізовані рекомендації.

PostgreSQL є надійною СУБД, однак для забезпечення максимальної безпеки необхідно вжити додаткових заходів [14]:

- використовувати надійні паролі: паролі повинні складатися не менше ніж з 12 символів і містити комбінацію літер, цифр і символів;
- увімкнути автентифікацію на основі пароля: це обов'язково для безпеки PostgreSQL;
- використовувати двофакторну аутентифікацію (2FA), що додає додатковий рівень безпеки базі даних;
- обмежити доступ до бази даних: лише люди, яким потрібен доступ до бази даних, повинні мати його;
- використовувати дозволи ролі: вони дозволяють призначати користувачам певні права доступу до бази даних;
- використовувати індекси, які можуть значно покращити продуктивність запитів;
- оптимізація запитів: використання ефективних запитів може значно підвищити продуктивність;
- використовувати реплікацію: це забезпечує резервне копіювання бази даних і високу доступність;
- використовувати кластеризацію, яка може значно підвищити продуктивність великих баз даних;

– використовувати масштабування: PostgreSQL підтримує масштабування, що дозволяє додавати додаткові сервери для підвищення продуктивності.

PostgreSQL займає почесне місце в розробці сучасних баз даних. Його важливість відображається в його надійності, продуктивності та різноманітних можливостях застосування. Якщо необхідно стати розробником баз даних, гарним вибором буде вивчення PostgreSQL. PostgreSQL — це потужна та гнучка СУБД.

2.4 Висновки за розділом 2

У другому розділі роботи вирішено актуальне завдання вибору та обґрунтування технічних засобів для реалізації проєкту. З метою створення ефективного вебзастосунку проведено аналіз доступних технологій, мов програмування та надбудов для організації процесу розробки. Дослідження переваг і недоліків різних рішень дозволило зробити обґрунтований вибір: для реалізації застосунку обрано мову програмування Python та фреймворк FastAPI, які забезпечують високу продуктивність, гнучкість і масштабованість системи.

3 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ ПРОГРАМИ

Система складається з двох частин: серверної та клієнтської, наведено на рисунку 3.1.

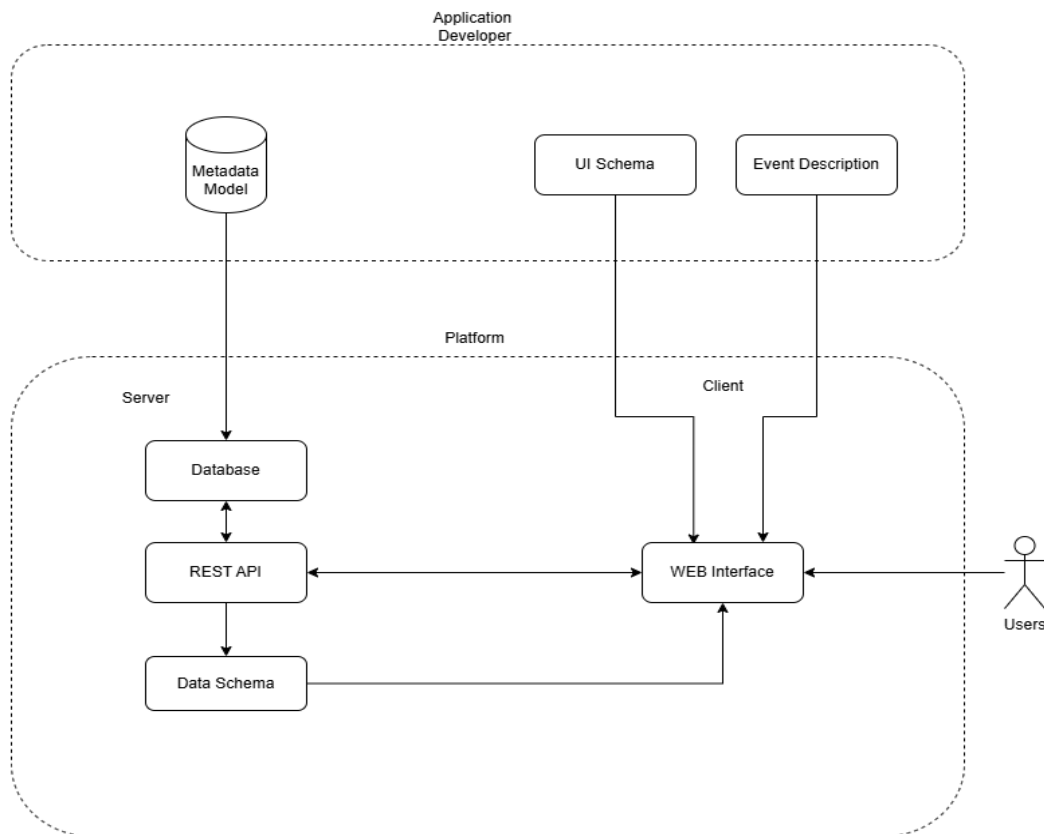


Рисунок 3.1 – Архітектура застосунку

3.1 Реалізація серверної частини

Функціональні вимоги серверної частини:

- генерування, по об'єктній моделі даних, структури бази даних;
- автоматична публікація REST API інтерфейс для роботи з даними;
- будування схеми даних для клієнта.

3.1.1 Автоматична генерація структури бази даних з об'єктної моделі

Об'єктно-реляційна модель дозволяє трансформувати структуру даних у вигляді об'єктів Python в структуру таблиць бази даних. Це досягається шляхом створення класів у вашому коді, які визначають, які поля міститиме таблиця, а також їхні типи даних, обмеження та зв'язки. Використовуючи ORM, такі класи можуть бути автоматично перетворені в оператори створення таблиці.

Визначення зв'язків між таблицями. Класи в Python можуть представляти реляційні зв'язки, такі як «один до багатьох» або «багато до багатьох». Це дозволяє моделювати складні системи, такі як CRM або системи обліку, де дані часто тісно пов'язані.

Автоматичне оновлення структури: коли структура класів змінюється (наприклад, коли додається нове поле), ORM дозволяє вносити ці зміни в базу даних через систему міграції. Це забезпечує зручний спосіб оновлення бази даних без втрати даних.

Переваги:

- розробнику не потрібно знати SQL для створення таблиць;
- швидке внесення змін у базу через редагування моделей;
- автоматична перевірка цілісності даних і зв'язків.

У системі low-code автоматизація структури бази даних дозволяє швидко створювати системи обліку або CRM без необхідності глибоко вникати в технічні деталі. Ця гнучкість важлива для адаптації системи до різних потреб.

У цій системі було розроблено мінімальну БД для облікової системи.

Було описано об'єктні моделі для таблиць «Customer», «Order», «OrderRow», «Item», «DeliveryMethod».

Сутність «Customer» служить для збереження даних про контрагента. Має в собі такі атрибути:

- id – ключовий атрибут;
- name – атрибут зберігає ім'я контрагента;
- number – атрибут зберігає номер контрагента;
- email – зберігає електронну пошту контрагента.

Сутність «DeliveryMethod» служить для збереження даних про метод доставки. Має в собі такі атрибути:

- id – ключовий атрибут;
- name – атрибут зберігає ім'я контрагента.

Сутність «Item» служить для збереження даних списку товарів . Має в собі такі атрибути:

- id – ключовий атрибут;
- name – атрибут зберігає ім'я контрагента.

Сутність «Order» служить для збереження властивостей замовлення. Має в собі такі атрибути:

- id – ключовий атрибут;
- number – атрибут зберігає номер замовлення;
- order_state – атрибут зберігає статус замовлення;
- customer – атрибут зв'язку між сутностями «Order» та «Customer»;
- delivery_method – атрибут зв'язку між сутностями «Order» та

«DeliveryMethod»;

- delivery_adress – атрибут зберігає дані про адрес доставки;
- waybill – атрибут зберігає дані про номер накладної;
- comment – атрибут зберігає коментарі до замовлення;
- date – атрибут зберігає дату замовлення.

Сутність «OrderRow» служить для збереження замовлення. Має в собі такі атрибути:

- id – ключовий атрибут;
- item – атрибут зв'язку між сутностями «Item» та «OrderRow»;
- quantity – атрибут зберігає кількість предмету товару;

- price – атрибут зберігає ціну за товар;
- sum – атрибут зберігає загальну суму за товар в залежності від вибраної кількості;
- order – атрибут зв'язку між сутностями «Order» та «OrderRow».

Структуру БД наведено на рисунку 3.2.

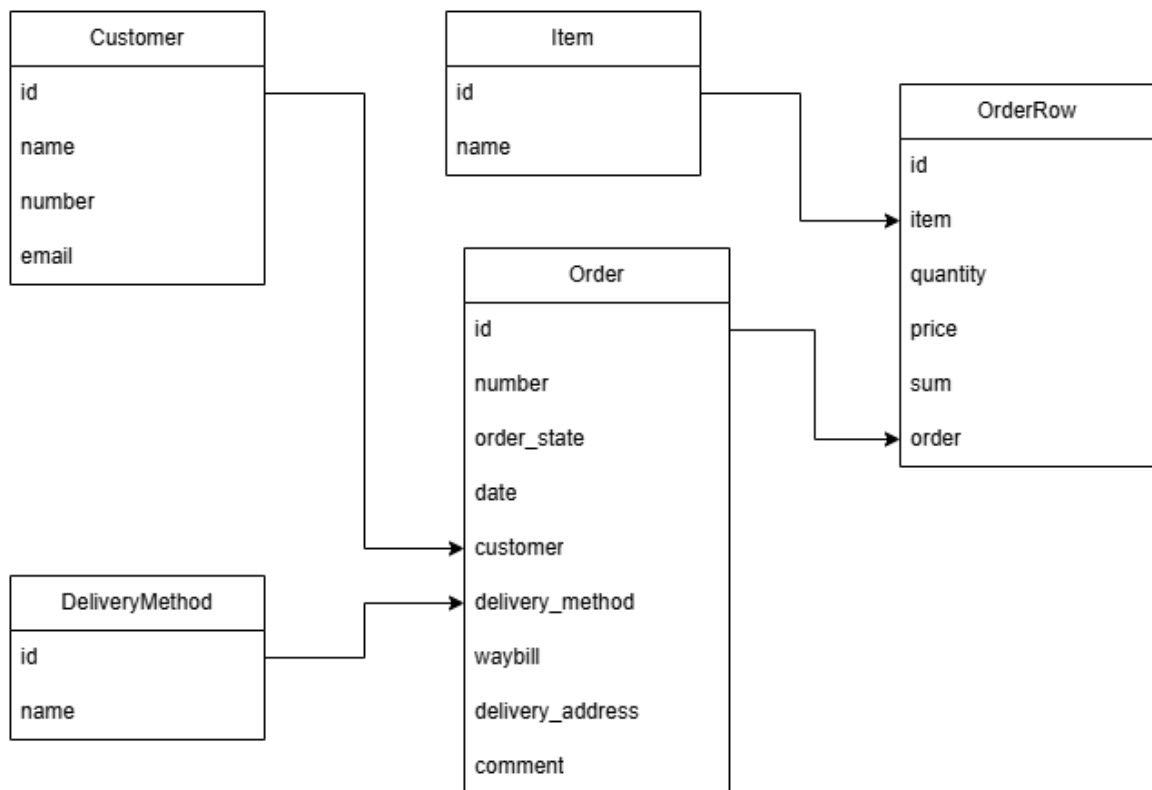


Рисунок 3.2 – Структура БД

3.1.2 Автономна публікація REST API для роботи з даними

REST (Representational State Transfer) API дозволяє програмам спілкуватися через запити HTTP. У low-code системах API створюється автоматично на основі моделі даних. Це значно скорочує час розробки та спрощує взаємодію з клієнтськими додатками.

Операції CRUD: основні операції автоматично генеруються для кожної сутності в системі:

- створення записів: до бази даних додаються нові об'єкти;
- читати записи: отримує дані за запитом;
- оновлення записів: існуючі дані змінено;
- видалити записи: об'єкти видаляються з бази даних.

Документація API. Система автоматично створює документацію OpenAPI з описом усіх доступних ендпоінтів, параметрів і типів відповідей. Це корисно для клієнтів, які інтегрують API у свої програми, оскільки вони можуть легко зрозуміти, як використовувати API.

Безпека: REST API може містити механізми автентифікації та авторизації для захисту даних. Це особливо важливо для систем CRM і систем обліку, де дані часто є конфіденційними.

У такій системі REST API надає стандартизований спосіб доступу до даних, особливо для побудови інтеграції з іншими системами. Це дозволяє клієнтам швидко підключатися до систем або CRM за допомогою попередньо створених ендпоінтів.

3.1.3 Генерація схеми даних для клієнтів

Схема даних описує структуру даних, що передаються між сервером і клієнтом. У контексті систем ця схема автоматично створюється на основі моделі даних і API.

Схеми JSON. Схеми JSON надають клієнтам структурований опис об'єктів, які вони отримують або надсилають через API. Наприклад, кожне поле описує його тип (рядок, число, об'єкт тощо), чи є поле обов'язковим та його можливі значення.

Інтеграція клієнта. Автоматично створені схеми JSON дозволяють клієнтам швидко налаштувати свої програми для роботи з вашою системою. Це особливо корисно для створення форм введення даних або перевірки на стороні клієнта.

Ваша система створює документацію OpenAPI, яка містить не лише список кінцевих точок, але й детальну інформацію про структуру даних. Це дозволяє розробникам легко використовувати API і навіть автоматично створювати клієнтські бібліотеки на основі цієї документації.

Створення схем даних у вашій системі полегшує процес інтеграції для користувачів. Оскільки графічного дизайнера немає, увага зосереджена на автоматизації документів і стандартизації структури даних, що дає розробникам більше гнучкості.

На рисунку 3.3 наведено приклад структури даних при виклику запиту GET для отримання списку даних з таблиці «Order».

```
{
  "id": 0,
  "date": "datetime",
  "number": "string",
  "order_state": {
    "id": 0,
    "name": "string"
  },
  "customer": {
    "id": 0,
    "name": "string",
    "mobile_number": "string",
    "email": "string",
    "comment": "string"
  },
  "mobile_number": "string",
  "delivery_method": {
    "id": 0,
    "name": "string"
  },
  "waybill": "string",
  "delivery_address": "string",
  "comment": "string",
  "orderrows": [
    {
      "id": 0,
      "row_index": 0,
      "item": {
        "id": 0,
        "name": "string",
        "vendor_code": "string",
        "deliverynoterows": [
          {
            "id": 0,
            "row_index": 0,
            "quantity": 0,
            "price": 0,
            "sum": 0,
            "delivery_note": {
              "id": 0,
              "date": "datetime",
              "number": "string"
            }
          }
        ]
      },
      "quantity": 0,
      "price": 0,
      "sum": 0
    }
  ]
}
```

Рисунок 3.3 – Структура даних таблиці «Order»

На рисунку 3.4 наведено приклад структури даних при виклику запиту PUT для оновлення даних в таблиці «OrderRow».

```
{
  "id": 0,
  "item": {
    "id": 0,
    "name": "string"
  },
  "quantity": 0,
  "price": 0,
  "sum": 0,
  "order": {
    "id": 0,
    "date": "datetime",
    "number": "string",
    "order_state": "string",
    "customer": {
      "id": 0,
      "name": "string",
      "mobile_number": "string",
      "email": "string"
    },
    "delivery_method": {
      "id": 0,
      "name": "string"
    },
    "waybill": "string",
    "delivery_address": "string",
    "comment": "string"
  }
}
```

Рисунок 3.4 – Структура даних таблиці «OrderRow»

3.2 Реалізація клієнтської частини

Функціональні вимоги клієнтської частини:

- конструює меню веб-додатку по схемі даних;
- за UI схемою відображає списки даних і форми даних;
- взаємодіє з серверною частиною через REST API інтерфейс.

3.2.1 Конструювання меню веб-додатку по схемі даних

На основі схем JSON або документації OpenAPI, отриманої з боку сервера, сторона клієнта автоматично генерує меню веб-додатку. Наприклад, схема може описувати назви розділів, підрозділів і доступні операції (створення, редагування, перегляд).

Схема даних аналізується для встановлення ієрархії розділів. Наприклад, якщо є категорії даних (клієнти, замовлення, товари), для кожної категорії створюється відповідний пункт меню.

У разі зміни серверної частини (додавання нової таблиці або функції) меню автоматично оновлюється без необхідності додаткового втручання в клієнтський код.

Переваги:

- автоматизація скорочує час на розробку;
- забезпечується відповідність між меню та функціональністю веб-додатку.

3.2.2 Відображення списків даних і форм за UI схемою

Схема інтерфейсу користувача: це спеціальний опис, який визначає вигляд списків і форм в інтерфейсі користувача. Наприклад:

- списки даних можуть містити стовпці, сортування та фільтри;
- форми можуть містити поля для введення даних, розкриті списки.

Система аналізує UI schema інтерфейсу користувача та автоматично створює елементи інтерфейсу.

Для кожного розділу меню створюється відповідний список або форма. Наприклад, для таблиці «Користувачі» система автоматично створює список із полями «Ім'я», «Контактний номер» і «Email».

Коли на серверній частині змінюється структура даних або додаються нові поля, UI schema дозволяє швидко оновлювати інтерфейс без необхідності вручну редагувати код.

Форми дозволяють вводити та редагувати дані. Перевірку можна виконати як на рівні клієнта, так і на рівні сервера, щоб переконатися, що введені значення правильні.

Переваги:

- мінімізуйте зусилля розробника для створення інтерфейсу;
- забезпечте однаковий вигляд усіх списків і форм.

3.2.3 Взаємодія з серверною частиною через REST API інтерфейс

Клієнтська частина використовує REST API для:

- читання даних: отримуйте списки або деталі про окремі об'єкти;
- запис даних: надсилання змін або нових об'єктів до бази даних сервера;
- видалити: видалити записи з відповідним запитом API.

Механізми взаємодії:

- автентифікація та авторизація: клієнт надсилає маркери або інші облікові дані для перевірки прав доступу;
- обробка помилок: наприклад, у разі помилки API клієнтська сторона відображає користувачеві повідомлення з поясненням проблеми (недійсні дані, заборона доступу тощо);
- асинхронність: запити API виконуються асинхронно, щоб забезпечити швидку відповідь інтерфейсу.

Переваги:

- гнучкість у взаємодії з даними;
- можливість інтеграції з іншими системами через REST API.

3.3 Висновок за розділом 3

У роботі досліджено актуальне завдання створення системи на основі low-code підходу. З метою розробки ефективного інструменту для побудови спеціалізованих додатків, таких як CRM або облікові системи, виконано фізичне та логічне проєктування вебзастосунку. Система забезпечує баланс між автоматизацією процесів і можливістю гнучкої модифікації вручну.

Клієнтська частина проєкту автоматизує взаємодію з даними та динамічно адаптується до структури, визначеної серверною частиною. Це дозволяє створювати простий, зручний інтерфейс для користувачів, який можна налаштувати відповідно до індивідуальних потреб підприємств. Таким чином, система поєднує автоматизацію, масштабованість і гнучкість, забезпечуючи адаптивність до змін і специфіки роботи різних організацій.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРЕМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення й умови застосування програми

Головним завданням роботи є дослідження швидкої розробки вебзастосунків та розробка конструктора на основі low-code підходу.

Low-code – це швидкий підхід до розробки, який дозволяє створювати програми та автоматизувати процеси за допомогою візуальних блоків, інтерфейсу перетягування, спадних меню та інших простих інструментів.

А інструменти інтеграції, такі як елементи plug-and-play, API, готові з'єднувачі та веб-сервіси, дозволяють легко підключати рішення з low-code до існуючих програм і систем.

Платформи low-code зазвичай не лише надають широкий спектр готових компонентів, але й сумісні з різними хмарними службами, алгоритмами штучного інтелекту, машинним навчанням і системами розпізнавання зображень, мови та облич.

Low-code дозволяє навіть нетехнічним спеціалістам створювати програми. Але крім того, це також корисно для розробників, оскільки вони можуть створювати складніші рішення набагато швидше та легше, ніж за допомогою традиційної розробки.[15]

Функціональні вимоги серверної частини:

- генерування, по об'єктній моделі даних, структури бази даних;
- автоматична публікація REST API інтерфейс для роботи з даними;
- будування схеми даних для клієнта.

Функціональні вимоги клієнтської частини:

- конструює меню веб-додатку по схемі даних;
- за UI схемою відображає списки даних і форми даних;
- взаємодіє з серверною частиною через REST API інтерфейс.

4.2 Умови виконання програми

Мінімальні системні вимоги та програмне забезпечення для функціонування серверної частини системи документообігу:

- продуктивний хостинг;
- вебсервер Gunicorn;
- підтримка Python та FastAPI.

Для експлуатації клієнтської частини необхідні такі програмно-технічні засоби:

- процесор із тактовою частотою від 2 GHz;
- комп'ютер з графічною ОС;
- оперативної пам'яті від 4GB;
- підключення до Internet мережі.

4.3 Розгортання застосунку

Структура вебзастосунка на основі використання обраного Framework FastApi.

Архітектура серверу, на якому розгортається застосунок:

- операційна система Linux;
- веб-сервер Gunicorn;
- сервер бази даних MySQL;
- версія Python 3.11.5.

4.4 Алгоритм послідовності взаємодії з системою

Першим кроком треба створити БД. На серверній частині у файлі «database.py» описуємо підключення до нашої БД.

Другим кроком буде розробка об'єктної моделі БД. Це робиться у каталозі «md_model». Описуємо моделі для кожної таблиці. Після цього за допомогою міграцій записуємо зміни у нашу БД.

Серверна частина автоматично генерує та публікує REST API інтерфейс, та будує схему даних.

Структура серверної частини наведена на рисунку 4.1

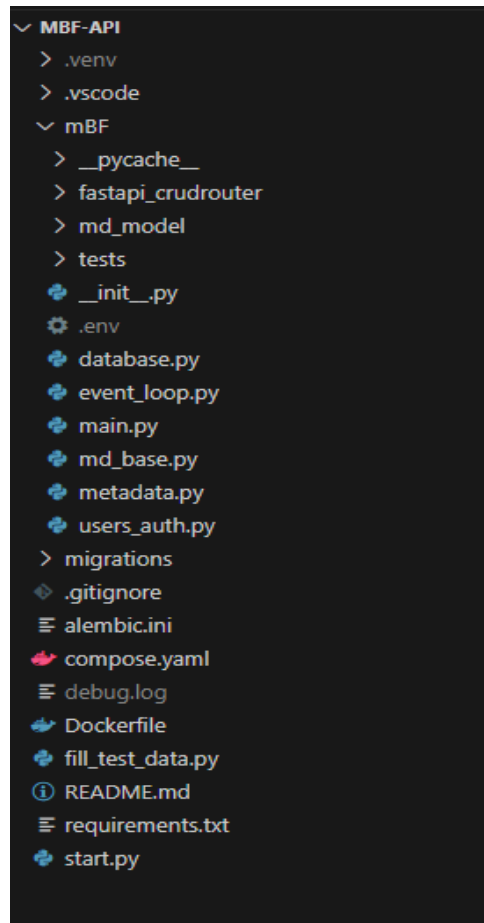


Рисунок 4.1 – Структура серверної частини

Далі в клієнтській частині вже описан арі запит, за допомогою якого він згенерує, за схемою даних, та відобразить меню вебзастосунку для клієнта у браузері.

При натисканні на пункти меню будуть відпрацьовувати арі запити для отримання даних з БД, які описані в файлі «events_metadata.py». За відображення даних відповідає файл «nicegui_json_schema_form.py». Він

відображає списки даних. Якщо вибрати якесь замовлення, щоб побачити форму з розгорнутою інформацією про нього, то до генерації форми ще додається файл «order.json», там описана структура для генерування форми. Користувач може редагувати його під потреби його додатку.

Також у файлі «events.py» описуються події, наприклад, для мінімального функціоналу системи обліку, яка була вибрана для тестування системи. В цей файл користувач може прописувати свої події, які потрібні для роботи з його продуктом. Приклади подій, які вже описані в системі:

- створення нового замовлення;
- редагування інформації в замовленні;
- збереження змін;
- видалення записів;
- генерація та скачування PDF документа з інформацією про замовлення, форма PDF документа(як дані будуть відображатися у файлі) описана у файлі «order_template.html», її користувач теж може переналаштувати під себе.

Структура клієнтської частини наведена на рисунку 4.2

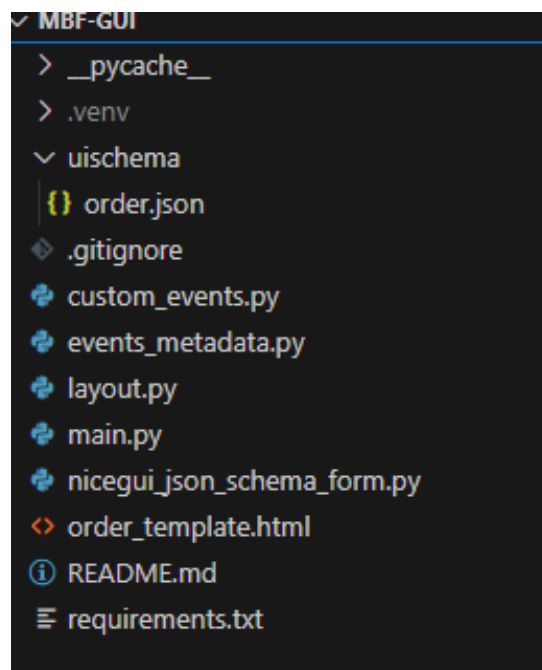


Рисунок 4.2 – Структура клієнтської частини

На рисунку 4.3 наведено алгоритм взаємодії з системою

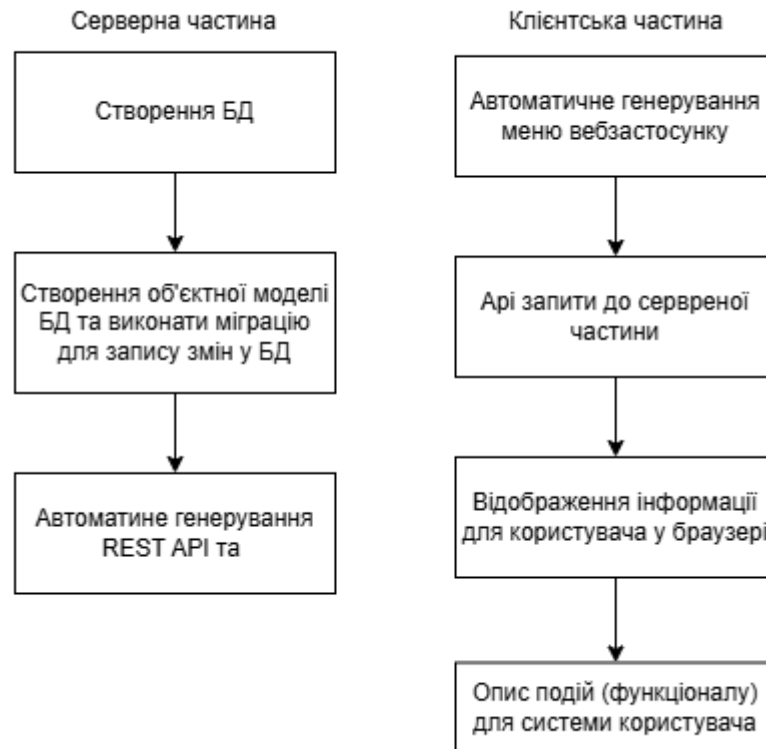


Рисунок 4.3 – Алгоритм взаємодії з системою

4.5 Звернення до програми

Звертання до програми від користувача організовано через відправлення запиту про перехід на сайт, використовуючи адресний рядок у веббраузері. Також звертання до програми відбувається, коли користувач вбудований функціонал системи, наприклад, роблячи API запит до сервера.

4.6 Методика та результат тестування

Для тестування системи було вбудовано мінімальну систему обліку. Було розроблено об'єктну модель.

Платформа автоматично генерує та публікує REST API інтерфейс на основі структури БД, що наведено на наступних рисунках 4.4 – 4.6.

Customer		^
GET	/customer Get All	▼
POST	/customer Create One	▼
DELETE	/customer Delete All	▼
GET	/customer/{item_id} Get One	▼
PUT	/customer/{item_id} Update One	▼
DELETE	/customer/{item_id} Delete One	▼

Рисунок 4.4 – Сгенерований REST API для Customer

Item		^
GET	/item Get All	▼
POST	/item Create One	▼
DELETE	/item Delete All	▼
GET	/item/{item_id} Get One	▼
PUT	/item/{item_id} Update One	▼
DELETE	/item/{item_id} Delete One	▼

Рисунок 4.5 – Сгенерований REST API для Item

Order		^
GET	/order Get All	▼
POST	/order Create One	▼
DELETE	/order Delete All	▼
GET	/order/{item_id} Get One	▼
PUT	/order/{item_id} Update One	▼
DELETE	/order/{item_id} Delete One	▼

Рисунок 4.6 – Сгенерований REST API для Order

Система сгенерував схему даних, клієнтська частина буде меню веб застосунку, наведена на рисунку 4.7.

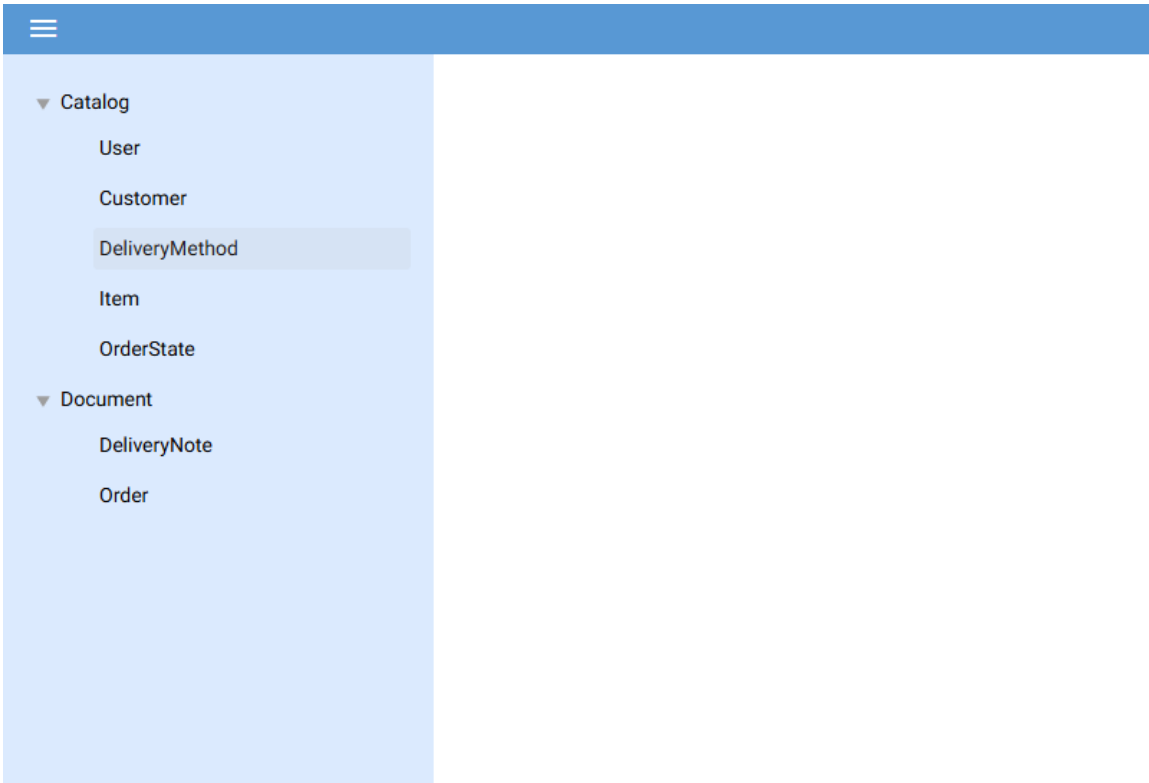


Рисунок 4.7 – Меню вебзастосунку

Наступним кроком тестували як за допомогою UI schema було відображено списки та форми даних.

На рис. 4.8 наведено список замовлень.

Order_SUK					
NEW ORDER					
Id	Date	Number	OrderState_ABY	Customer_UWY	Mobile Number
1	2023-12-16T14:50:50	TST-000001	{ "id": 1 }	{ "id": 1 }	+380661122334
2	2023-12-17T11:32:53.199448	TST-000002	{ "id": 2 }	{ "id": 2 }	+380675556677
3	2024-01-25T13:29:47	TST-000003	{ "id": 3 }	{ "id": 2 }	+38094181238
SUBMIT					

Рисунок 4.8 – Список замовлень

На рисунку 4.7 наведено список каталогу.

Name	Vendor Code
Процесор INTEL S1700 Core i5-12600	int12600
Мат. плата 1700 ASUS PRIME Z790-P	asusZ790P
Відеокарта Gigabyte GeForce RTX 4060 Ti 8Gb	gb4060ti8

Рисунок 4.7 – Список каталогу

На рисунку 4.8 наведено список контрагентів

Customer_UWY

NEW ORDER

Id	Name	Mobile Number
1	Иванов	+380661122334
2	Петров	+380675556677

Рисунок 4.8 – Список контрагентів

Далі на рисунку 4.9 наведено форму даних обранного замовлення, яка побудована за допомогою UI схемою.

Id

1

Date

2023-12-16T14:50:50

Number

TST-000001

Order state

Новий

Delivery method

Новим листом

Customer

Иванов

Delivery address

Одеса спр-ення 35

Mobile #

+380661122334

Waybill

2005 3456 5678 7890

Comment

ADD

DELETE

DOWNLOAD

☐	Row Index	Item	Quantity	Price	Sum
☐	1	Процесор INTEL S1700 Core i5-12600	2	10200	20400
☐	2	Відеокарта Gigabyte GeForce RTX 4060 Ti 8Gb	2	9000	18000
Total			4		38400

SAVE

Рисунок 4.9 – Форма даних замовлення

Також було перевірено роботу арі за допомогою функцій редагування, видалення інформації та додавання нового запису у БД.

ВИСНОВКИ

У роботі було досліджено актуальне завдання створення конструктора для швидкої розробки вебзастосунків на основі low-code підходу. Основною метою стало вивчення особливостей даного підходу та розробка функціональної системи, що відповідала б сучасним вимогам автоматизації розробки.

Для цього було виконано ряд завдань:

- проведено аналіз переваг і недоліків low-code підходу;
- сформовано список функціональних і технічних вимог до системи;
- виконано детальне проєктування структури вебзастосунку;
- реалізовано програмну частину та проведено тестування.

Структура вебзастосунку була реалізована із використанням Python з Framework FastAPI.

Був розроблений сценарій та проведено тестування вебзастосунку. Сценарій тестування був на основі основної логіки використання системи. Отриманий результат свідчить про позитивний результат.

В результаті виконання дипломної роботи було отримано low-code систему, що відповідає всім поставленим вимогам, які були визначені на початку розробки та закріпленим в технічному завданні. Розроблена система вийшла зручною для використання користувачем та виконує всі поставлені функції, які були визначені на початку роботи .

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Навіщо потрібні low-code платформи? [Електрон. ресурс]. – Режим доступу: <https://itedu.center/ua/blog/articles/zachem-nuzhny-low-code-platformy>.
2. Що таке low-code та no-code платформи та чи загрожують вони традиційній розробці? [Електрон. ресурс]. – Режим доступу: <https://proit.ua/shcho-takie-low-code-ta-no-code-platformi-ta-chi-zaghrozhuut-voni-traditsiinii-rozrobsi>.
3. Чому low-code платформи – це майбутнє розробки програмного забезпечення? [Електрон. ресурс]. – Режим доступу: <https://scriptum.ua/chomu-low-code-platformy-cze-majbutnye-rozrobky-programnogo-zabezpechennya>.
4. Секрети успішного впровадження low-code у великих компаніях [Електрон. ресурс]. – Режим доступу: <https://scriptum.ua/low-code-implementation>.
5. Причини для впровадження low/no-code у бізнес: поради та практичне застосування [Електрон. ресурс]. – Режим доступу: <https://brandstory.com.ua/kejsy/pricini-dla-vprovadzenna-low-no-code-u-biznes-poradi-ta-prakticne-zastosuvanna>.
6. Поширені міфи про low-code: вигадка чи реальність [Електрон. ресурс]. – Режим доступу: <https://scriptum.ua/poшыreni-mify-pro-low-code-vygadka-chy-realnist>.
7. Low-code integration with everything [Electronic resource]. – Access mode: [//www.outsystems.com/low-code-platform/integration](https://www.outsystems.com/low-code-platform/integration).
8. Everything you need to know about Mendix [Electronic resource]. – Access mode: <https://bizzomate.com/en/blog/everything-you-need-to-know-about-mendix-find-out-what-it-is-and-how-it-works>.
9. Чому варто обрати Python [Електрон. ресурс]. – Режим доступу: <https://wezom.academy.ua/pochemu-stoit-vybrat-python>.

10. Переваги і недоліки мови Python [Електрон. ресурс]. – Режим доступу: <https://blog.ithillel.ua/articles/perevagi-i-nedoliki-movi-python>.
11. Framework FastAPI [Electronic resource]. – Access mode: <https://fastapi.tiangolo.com/uk>.
12. FastAPI vs Flask: Choosing the Right Web Development Framework [Electronic resource]. – Access mode: <https://shakuro.com/blog/fastapi-vs-flask>.
13. Python FastAPI vs Flask: A Detailed Comparison [Electronic resource]. – Access mode: <https://www.turing.com/kb/fastapi-vs-flask-a-detailed-comparison>.
14. Що таке PostgreSQL і для чого використовується [Електрон. ресурс]. – Режим доступу: <https://foxminded.ua/postgresql-shcho-tse>.
15. Різниця між low-code та no-code: порівняння та можливості кожної технології [Електрон. ресурс]. – Режим доступу: <https://scriptum.ua/riznyczya-mizh-low-code-ta-no-code-porivnyannya-ta-mozhlyvosti-kozhnoyi-tehnologiyi>.

ДОДАТОК А
Технічне завдання

Вступ

Область застосування програмного забезпечення – будь-яке підприємство, що потребує, швидкої та з мінімальними витратами, розробки спеціалізованих ПЗ, вручну додаючи необхідні функції.

Тема для проектування системи – дослідження швидкої розробки вебзастосунків та розробка конструктора на основі low-code підходу.

A.1 Підстава для розробки

Підставою для виконання роботи є завдання на дипломну кваліфікаційну роботу на тему «Дослідження швидкої розробки вебзастосунків та розробка конструктора на основі low-code підходу» затверджене наказом Національного університету «Запорізька політехніка», протокол № 398 від 7 жовтня 2024 року.

A.2 Основні вимоги до програми, що розробляється

Головним завданням роботи є створення системи, яка дозволяє користувачу швидко та з мінімальними затратами розробити CRM або облікову систему.

A.2.1 Вимоги до функціональних характеристик

Функціональні вимоги серверної частини:

- генерування, по об'єктній моделі даних, структури бази даних;
- автоматична публікація REST API інтерфейс для роботи з даними;
- будівництва схеми даних для клієнта.

Функціональні вимоги клієнтської частини:

- конструює меню веб-додатку по схемі даних;

- за UI схемою відображає списки даних і форми даних;
- взаємодіє з серверною частиною через REST API інтерфейс.

A.2.2 Експлуатаційні вимоги

Для експлуатації клієнтської частини необхідні такі програмно-технічні засоби:

- процесор із тактовою частотою від 2 GHz;
- комп'ютер з графічною ОС;
- оперативної пам'яті від 4GB;
- підключення до Internet мережі.

Не функціональні вимоги до продукту наступні:

- продуктивний хостинг;
- вебсервер Gunicorn;
- підтримка Python та FastAPI.

Для експлуатації серверної частини необхідні такі програмно-технічні засоби:

- процесор із тактовою частотою від 2 GHz;
- комп'ютер з графічною ОС;
- оперативної пам'яті від 4GB;
- підключення до Internet мережі.

A.3 Вхідні дані до роботи

В якості вхідних даних виступають дані, які надходять від віддаленого клієнта про запитувану сторінку або про виконання деяких дій, а вихідними даними є результати обробки запитів клієнта і формування вебсторінки.

ДОДАТОК Б
Текст програми

Б.1 Перший файл main.py (Серверна частина)

```

from fastapi import FastAPI, APIRouter, Request, Response, Depends
from .fastapi_crudrouter import OrmarCRUDRouter
from fastapi.middleware.cors import CORSMiddleware
from starlette.types import Message
from starlette.background import BackgroundTask
from typing import List
import logging
from fastapi.security import OAuth2PasswordRequestForm

from . import metadata
from .database import database
from . import event_loop

app = FastAPI()
logging.basicConfig(filename='debug.log', level=logging.DEBUG)

origins = ["*"]

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

def log_info(req_headers, req_body, res_body):
    logging.info(req_headers)
    logging.info(req_body)
    logging.info(res_body)

async def set_body(request: Request, body: bytes):
    async def receive() -> Message:

```

```

        return {'type': 'http.request', 'body': body}
    request._receive = receive

@app.middleware('http')
async def some_middleware(request: Request, call_next):
    req_body = await request.body()
    await set_body(request, req_body)
    response = await call_next(request)

    res_body = b""
    async for chunk in response.body_iterator:
        res_body += chunk

    task = BackgroundTask(log_info, dict(request.headers), req_body, res_body)
    return Response(content=res_body, status_code=response.status_code,
                    headers=dict(response.headers), media_type=response.media_type,
background=task)

@app.on_event("startup")
async def startup():
    await database.connect()
    event_loop.get_event_loop()

@app.on_event("shutdown")
async def shutdown():
    await database.disconnect()

routers = []

schema_router = APIRouter(prefix='/schema')

for currentClass in metadata.get_all_classes():
    try:
        if currentClass.__useInAPI__:
            routers.append(

```

```

OrmarCRUDRouter(
    schema=currentClass,
    prefix=currentClass.__apiPrefix__,
)

currentSchema = metadata.SchemaModel(currentClass)
@schema_router.get("/"+currentClass.__apiPrefix__)
def get_object_schema(model = Depends(currentSchema)):
    return model

except:
    pass

#@routerOrder.get("", response_model=List[metadata.OrderView])
#def overloaded_get_all(db: Session = Depends(get_db)):
#    orders = db.exec(select(metadata.Order)).all()
#    return orders

for router in routers:
    app.include_router(router)

app.include_router(schema_router)
@app.get("/menu", response_model=List[metadata.Menu])
def get_menu():
    menu = []
    return metadata.get_menu_tree(metadata.Metadata, menu=menu)

```

Б.2 Другой файл md_base.py (Серверная часть)

```

from datetime import datetime
from typing import Optional, List
import ormar
import sqlalchemy
from pydantic import BaseModel

from .database import database

```

```

from mBF.database import engine

metadata = sqlalchemy.MetaData()

def getDocumentNumber(table_name):
    with engine.connect() as con:
        try:
            rs = con.execute(f'SELECT number FROM {table_name} ORDER BY number
DESC LIMIT 1')
            for row in rs:
                max_number = (row[0])
                prefix, delimetr, number = max_number.partition("-")
                next_number = str(int(number)+1).zfill(len(number))
                return prefix + delimetr + next_number
        except ValueError:
            return "TST-000001"
        except:
            return "TST-000001"

class BaseMeta(ormar.ModelMeta):
    metadata = metadata
    database = database

class Menu(BaseModel):
    id: str
    label: str
    children: List['Menu'] = []

class Metadata(ormar.Model):
    class Meta:
        abstract = True
        __useInMenu__ = False
        __useInAPI__ = False
        __apiPrefix__ = None
    id: int = ormar.Integer(primary_key=True)

```

```

class Catalog(Metadata):
    class Meta:
        abstract = True
        __useInMenu__ = True
        name: str = ormar.String(max_length=500)

class Document(Metadata):
    class Meta:
        abstract = True
        __useInMenu__ = True
        date: Optional[datetime] = ormar.DateTime(default=datetime.now)
        number: Optional[str] = ormar.String(max_length=20)

class RowBase(Metadata):
    class Meta:
        abstract = True
        row_index: Optional[int] = ormar.Integer()

# Users table shema
class Token(BaseModel):
    access_token: str
    token_type: str

class TokenData(BaseModel):
    username: str | None = None

class User(Catalog):
    class Meta(BaseMeta):
        pass
        __useInAPI__ = True
        __apiPrefix__ = "user"
        username: str = ormar.String(max_length=50)
        email: str = ormar.String(max_length=50)
        full_name: str = ormar.String(max_length=100)
        disabled: bool = ormar.Boolean(default=False)
        hashed_password: str = ormar.String(max_length=100)

```

Б.3 Третій файл order.json (Клієнтська частина)

```
{
  "type": "VerticalLayout",
  "label": "Order",
  "elements": [
    {
      "type": "Control",
      "label": "Id",
      "scope": "#/properties/id",
      "options": {
        "visible": false
      }
    },
    {
      "type": "Control",
      "label": "Date",
      "scope": "#/properties/date"
    },
    {
      "type": "Control",
      "label": "Number",
      "scope": "#/properties/number"
    },
    {
      "type": "HorizontalLayout",
      "elements": [
        {
          "type": "VerticalLayout",
          "label": "Customer info",
          "elements": [
            {
              "type": "Control",
              "label": "Order state",
              "scope": "#/properties/order_state"
            }
          ]
        }
      ]
    }
  ]
}
```

```

    },
    {
      "type": "Control",
      "label": "Customer",
      "scope": "#/properties/customer"
    },
    {
      "type": "Control",
      "label": "Mobile #",
      "scope": "#/properties/mobile_number"
    }
  ]
},
{
  "type": "VerticalLayout",
  "label": "Delivery info",
  "elements": [
    {
      "type": "Control",
      "label": "Delivery method",
      "scope": "#/properties/delivery_method"
    },
    {
      "type": "Control",
      "label": "Delivery address",
      "scope": "#/properties/delivery_address"
    },
    {
      "type": "Control",
      "label": "Waybill",
      "scope": "#/properties/waybill"
    }
  ]
}
]

```

```

},
{
  "type": "Control",
  "label": "Comment",
  "scope": "#/properties/comment"
},
{
  "type": "Table",
  "label": "Orderrows",
  "scope": "#/properties/orderrows",
  "elements": [
    {
      "type": "Control",
      "label": "Id",
      "scope": "#/properties/orderrows/items/properties/id",
      "options": {
        "visible": false
      }
    },
    {
      "type": "Control",
      "label": "Row Index",
      "scope": "#/properties/orderrows/items/properties/row_index"
    },
    {
      "type": "Control",
      "label": "Item",
      "scope": "#/properties/orderrows/items/properties/item",
      "options": {
        "format": "dropdownlist"
      }
    },
    {
      "type": "Control",
      "label": "Quantity",

```

```

        "hasTotal": true,
        "scope": "#/properties/orderrows/items/properties/quantity"
    },
    {
        "type": "Control",
        "label": "Price",
        "scope": "#/properties/orderrows/items/properties/price"
    },
    {
        "type": "Control",
        "label": "Sum",
        "hasTotal": true,
        "scope": "#/properties/orderrows/items/properties/sum"
    }
]
}
]
}

```

Б.4 Четвертий файл layout.py(Клієнтська частина)

```

from nicegui import events, ui
import requests
import jsonref
import json

import nicegui_json_schema_form

class MainLayout:

    def __init__(self):
        self.uiObjects = []
        self.uiHeader = ui.header().classes(replace='row items-center')
        self.uiSideMenu = ui.left_drawer().classes('bg-blue-100')
        self.uiFooter = ui.footer(value=False)

```

```

self.uiSticky = ui.page_sticky(position='bottom-right', x_offset=20, y_offset=20)
self.tabs = []
self.Form = []

with self.uiHeader:
    ui.button(on_click=lambda: self.uiSideMenu.toggle(), icon='menu').props('flat
color=white')
    self.uiTabs = ui.tabs()

with self.uiSideMenu:
    menu = requests.get('http://127.0.0.1:8000/menu').json()

    self.uiTree = ui.tree(menu, on_select=self.menuEvent)

    self.uiTree.props(add='no-connectors')
    self.uiTree.props(add='selected-color="primary"')

self.uiPanels = ui.tab_panels(self.uiTabs).classes('w-full')

with self.uiSticky:
    ui.button(on_click=self.uiFooter.toggle, icon='contact_support').props('fab')

with self.uiFooter:
    ui.label('Footer')

def addTabObject(self, name, render_data, render_schema, render_uischema):
    with self.uiHeader:
        with self.uiTabs:
            if not any(el._props["name"] == name for el in self.tabs):
                self.tabs.append(ui.tab(name))

        with self.uiPanels:
            with ui.tab_panel(name):
                form = nicegui_json_schema_form.RenderForm(self, render_data,
render_schema, render_uischema)

```

```

        form.render_form()
        self.uiPanels.set_value(name)

def addTabListOfObject(self, name, render_data, render_schema):
    with self.uiHeader:
        with self.uiTabs:
            if not any(el._props["name"] == name for el in self.tabs):
                self.tabs.append(ui.tab(name))

        with self.uiPanels:
            with ui.tab_panel(name):
                form = nicegui_json_schema_form.RenderList(self, render_data,
render_schema)

                form.form_from_json_schema(render_data, render_schema)
                ui.button('Submit', on_click=self.getFormData)
            self.uiPanels.set_value(name)

def menuEvent(self, e: events.GenericEventArguments) -> None:
    if e.value != None:
        selected_menu = e.value.lower()
        schema_response = requests.get('http://127.0.0.1:8000/schema/' +
selected_menu)
        object_response = requests.get('http://127.0.0.1:8000/' + selected_menu)

        schema = jsonref.loads(schema_response.text)
        object = object_response.json()
        self.addTabListOfObject(selected_menu, object, schema)

```

Б.5 П'ятий файл custom_events.py(Клієнтська частина)

```

from events_metadata import runner
@runner.add_custom_event('change:table:quantity')
def some_function(e_obj, e):
    OrderTableRecalc(e_obj, e)

```

```

@runner.add_custom_event('change:table:price')
def some_function(e_obj, e):
    OrderTableRecalc(e_obj, e)

def OrderTableRecalc(event_object, e):
    prop = event_object.event.replace('change:table:', '')
    for row in event_object.uiObject.rows:
        if row['row_index'] == e.args['row_index']:
            row[prop] = e.args[prop]

    if prop in ['quantity', 'price']:
        for row in event_object.uiObject.rows:
            if row['row_index'] == e.args['row_index']:
                row['sum'] = float(row['quantity']) * float(row['price'])
    event_object.uiObject.update()

```

ДОДАТОК В
Слайди презентації

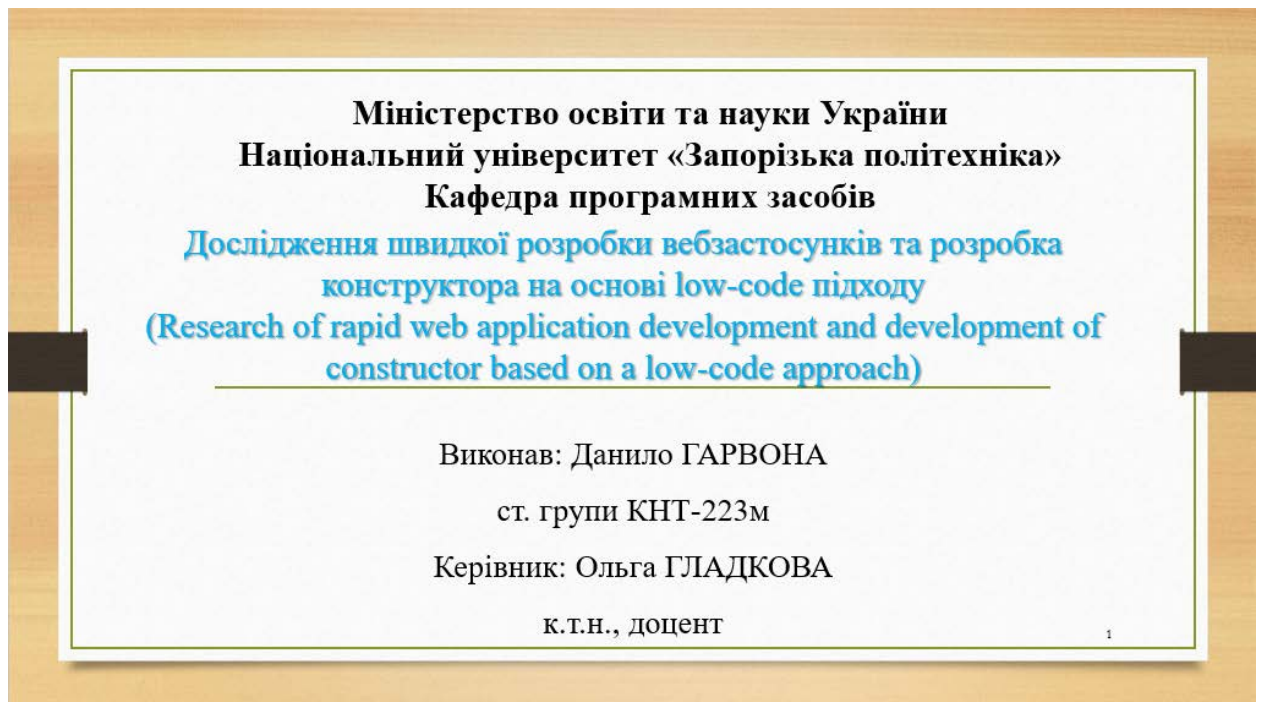


Рисунок В.1 – Слайд 1

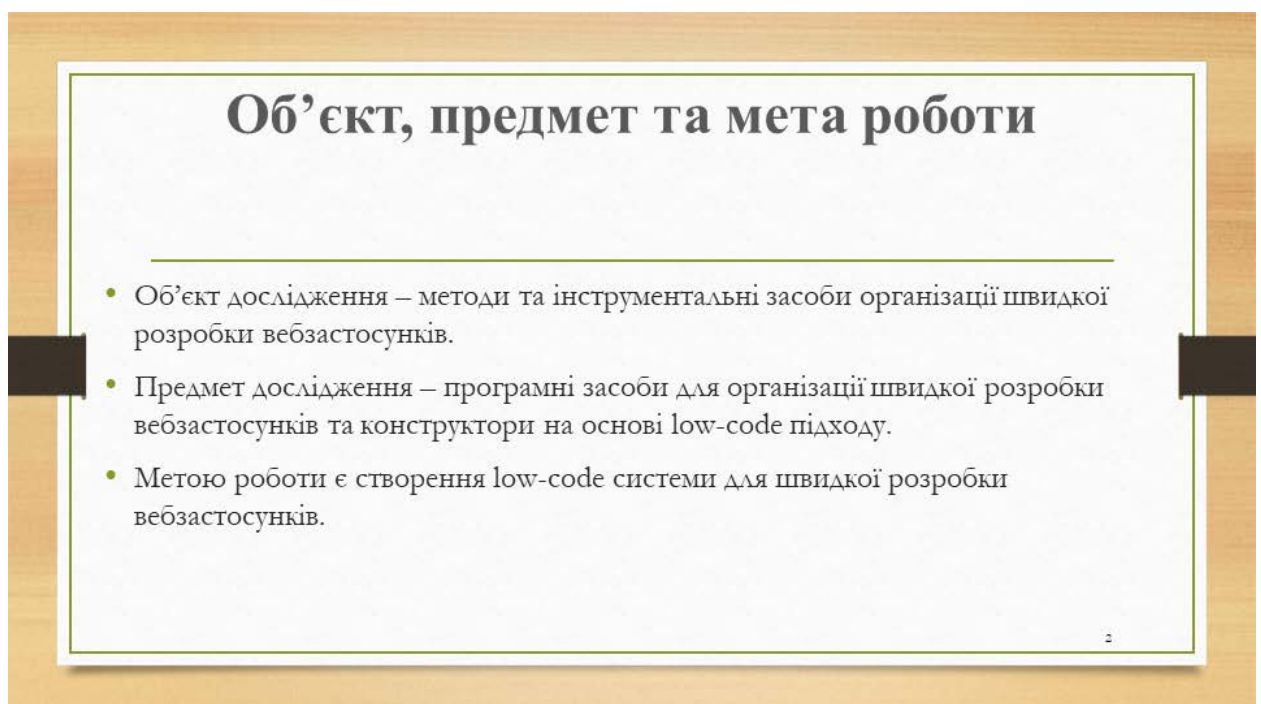


Рисунок В.2 – Слайд 2

Порівняння існуючих аналогів

Критерій	OutSystems	Mendix	Розроблюваний продукт
Безкоштовний	+	+	+
Графічний інтерфейс	+	+	-
Вимоги до системи	+	+	+
Швидкість навчання роботи	+	+	+
Персоналізована розробка під конкретні потреби бізнесу(CRM, системи обліку)	-	-	+

3

Рисунок В.3 – Слайд 3

Порівняння мов програмування

Критерій	PHP	Python	ASP.NET
Надбудови для структуризації розробки	Багато	Багато	Мало
Серверна підтримка	Розповсюджена	Розповсюджена	Недостатньо розповсюджена
Масштабованість розробок	Проста	Проста	Проста
Складність підтримки	Низька	Середня	Середня

5

Рисунок В.4 – Слайд 4

Порівняння Framework		
Критерії	<u>FastAPI</u>	<u>Flask</u>
Мова програмування	Python	Python
HTTP methods	@app.get("/")	@app.route("/", methods=["GET"])
Відображення помилки	Помилка відображається в JSON форматі	Помилка відображається на html сторінці
Перевірка даних	Вбудована функція перевірки даних	-
Підтримка асинхронних завдань	+	-
Продуктивність	Реалізує специфікації ASGI	Обмеження в програмах WSGI
<u>Підтримка</u>	<u>Менша підтримка</u>	<u>Більша підтримка</u>

Рисунок В.5 – Слайд 5



Рисунок В.6 – Слайд 6

Структура бази даних

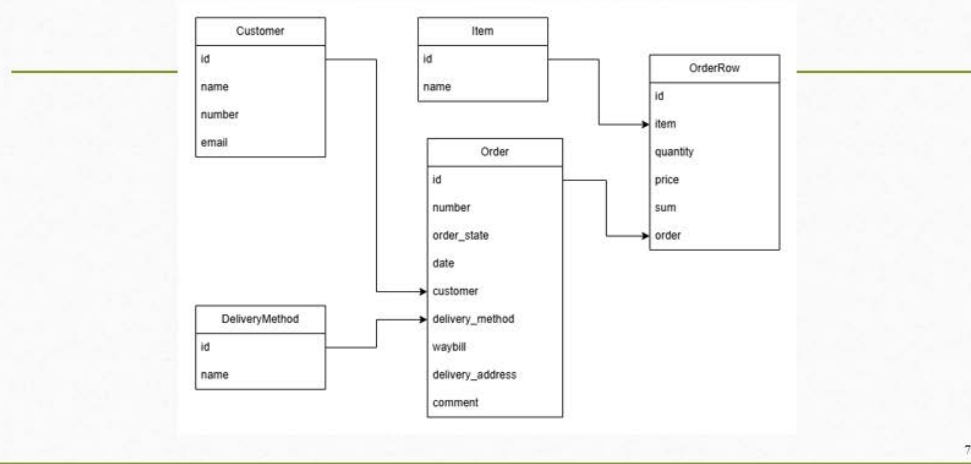


Рисунок В.7 – Слайд 7

Структури запитів

```

{
  "id": 0,
  "date": "datetime",
  "number": "string",
  "order_state": "string",
  "customer": {
    "id": 0,
    "name": "string",
    "mobile_number": "string",
    "email": "string",
    "comment": "string"
  },
  "delivery_method": {
    "id": 0,
    "name": "string"
  },
  "waybill": "string",
  "delivery_address": "string",
  "comment": "string",
  "orderrows": [
    {
      "id": 0,
      "item": {
        "id": 0,
        "name": "string",
        "mobile_number": "string",
        "email": "string",
        "comment": "string"
      },
      "quantity": 0,
      "price": 0,
      "sum": 0,
      "delivery_date": {
        "id": 0,
        "date": "datetime",
        "number": "string"
      }
    }
  ]
}
  
```

```

{
  "id": 0,
  "item": {
    "id": 0,
    "name": "string"
  },
  "quantity": 0,
  "price": 0,
  "sum": 0,
  "order": {
    "id": 0,
    "date": "datetime",
    "number": "string",
    "order_state": "string",
    "customer": {
      "id": 0,
      "name": "string",
      "mobile_number": "string",
      "email": "string"
    },
    "delivery_method": {
      "id": 0,
      "name": "string"
    },
    "waybill": "string",
    "delivery_address": "string",
    "comment": "string"
  }
}
  
```

Рисунок В.8 – Слайд 8

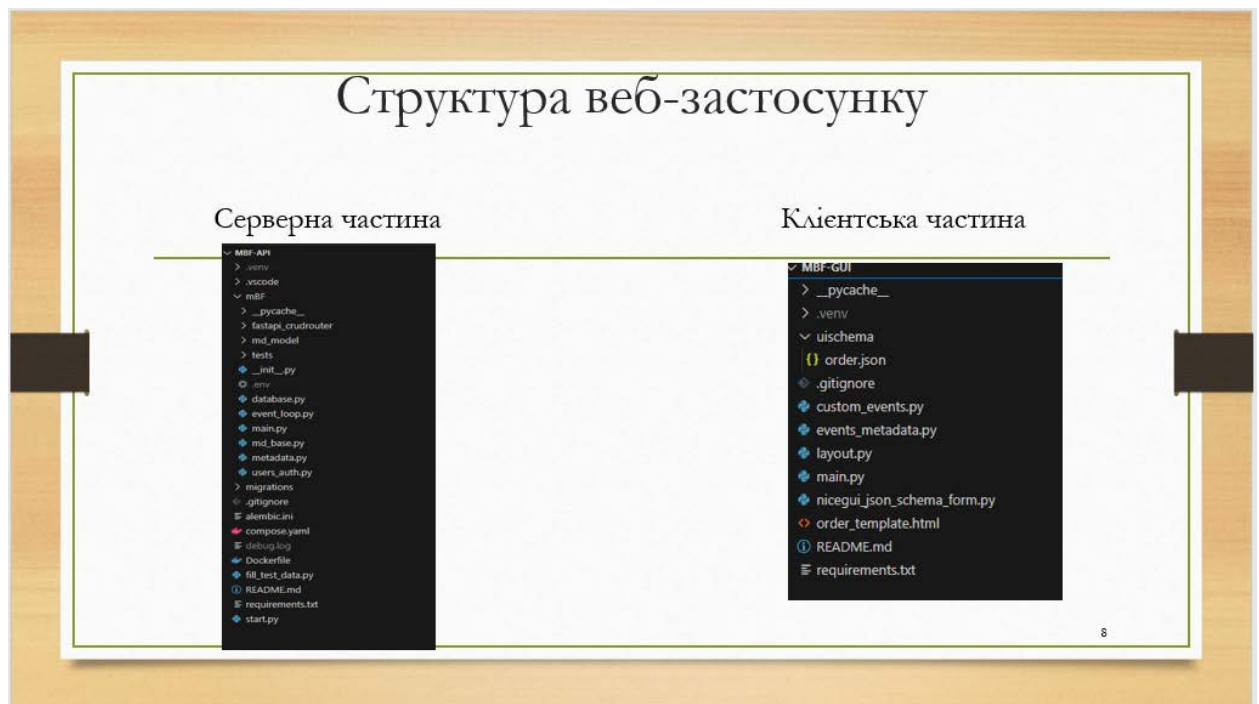


Рисунок В.9 – Слайд 9

Тестування застосунку

Під час тестування системи було вбудовано мінімальну систему обліку

- Було розроблено об'єктну модель;
- Системою було автоматично згенеровано REST API інтерфейс на основі структури БД та згенеровано структуру схеми даних.
- На клієнтській частині було отримано веб інтерфейс для користувача.
- За допомогою арі запитів отримано дані з бази даних.

Приклади на наступних слайдах

10

Рисунок В.10 – Слайд 10



Рисунок В.11 – Слайд 11

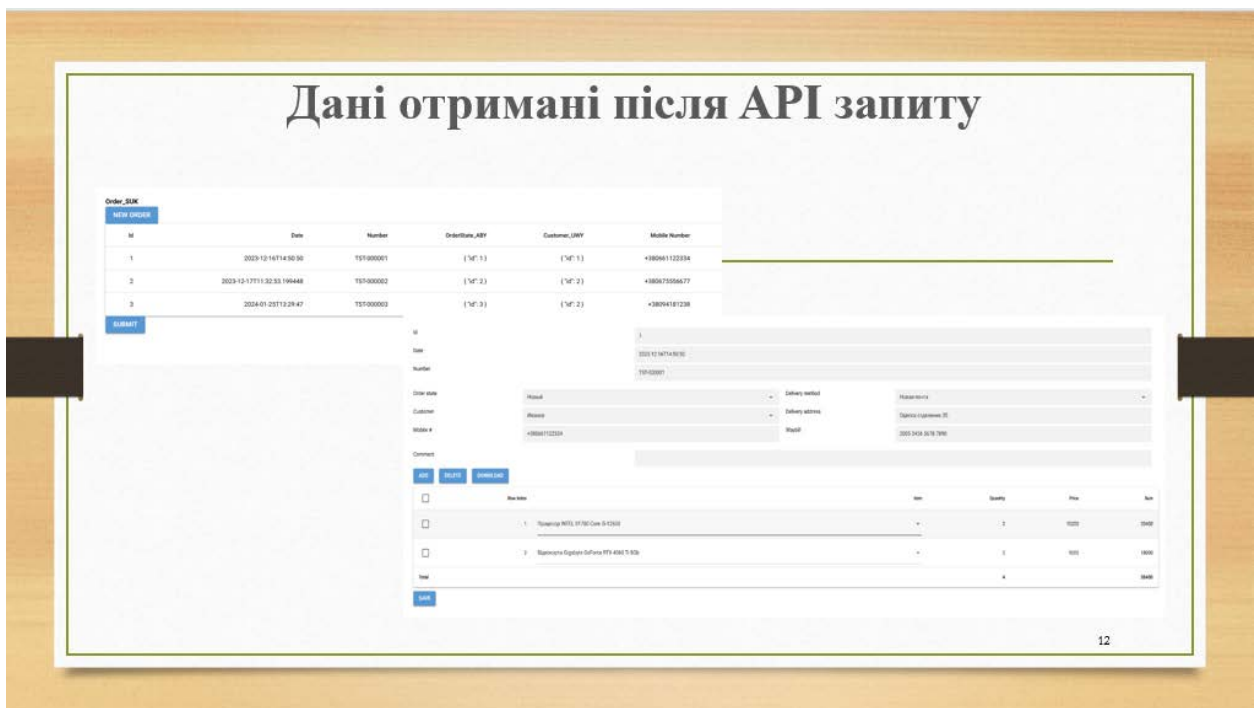


Рисунок В.12 – Слайд 12

Висновки

У роботі було досліджено актуальне завдання створення конструктора для швидкої розробки вебзастосунків на основі low-code підходу. Основною метою стало вивчення особливостей даного підходу та розробка функціональної системи, що відповідає б сучасним вимогам автоматизації розробки.

Для цього було виконано ряд завдань:

- проведено аналіз переваг і недоліків low-code підходу;
- сформовано список функціональних і технічних вимог до системи;
- виконано детальне проектування структури вебзастосунку;
- реалізовано програмну частину та проведено тестування.

В результаті виконання дипломної роботи було отримано low-code систему, що відповідає всім поставленим вимогам, які були визначені на початку розробки та закріпленим в технічному завданні. Розроблена система вийшла зручною для використання користувачем та виконує всі необхідні функції для розробки вебзастосунку.

13

Рисунок В.13 – Слайд 13