Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Волошко Максима Миколайовича*
(ПІБ)

академічної групи *121-19-2*
(шифр)

спеціальності *121 Інженерія програмного забезпечення*
(код і назва спеціальності)

освітньої програми *Інженерія програмного забезпечення*
(назва освітньої програми)

на тему: *Розробка веб-додатка взаємодії з базою даних MySQL за допомогою мови програмування Python та фреймворку fastAPI*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*Проф. Бердник М.Г.*			
розділів:				
спеціальний	*Проф. Бердник М.Г.*			
економічний	*доц. Касьяненко Л.В.*			
Рецензент				
Нормоконтролер	*ст.викл. Мартиненко А.А.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)

_____ М.О. Алексєєв
(підпис) (прізвище, ініціали)

« » _____ 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-19-2* *Волошко Максима Миколайовича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка веб-додатка взаємодії з Базою даних MySQL за допомогою мови програмування Python та фреймворку fastAPI*

затверджена наказом ректора НТУ «ДП» від *16.05.2023* № *350-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2023 р.***

Завдання видав _____ *Проф. Бердник М.Г.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Волошко М.М.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *12.06.2023 р.*

РЕФЕРАТ

Пояснювальна записка: 45с., 21 рис., 3 дод., 20 джерел.

Об'єкт розробки: веб-інтерфейс з доступом до взаємодії з базою даних.

Мета кваліфікаційної роботи: розробка веб-інтерфейсу за допомогою FasAPI у мові програмування python з документацією для розробників.

У вступній частині кваліфікаційної роботи проведено аналіз та розглянуто сучасний стан проблеми, визначено конкретну мету дослідження та галузь її застосування, наведено обґрунтування актуальності обраної теми та деталізовано постановку задачі.

У першому розділі кваліфікаційної роботи було проведено детальний аналіз предметної галузі, розглянуто актуальність поставленої задачі та цілей розробки, сформульовано конкретну постановку завдання, а також визначено вимоги до програмної реалізації, використовуваних технологій та програмних засобів.

У другому розділі кваліфікаційної роботи було проаналізовано наявні рішення, обрані платформи для розробки, проведено проектування та розробка програми. Також була надана детальна інформація про роботу програми, включаючи опис алгоритму та структури її функціонування. В розділі було розглянуто процес виклику та завантаження програми, визначено вхідні та вихідні дані, а також надано характеристику параметрів технічних засобів.

У економічному розділі кваліфікаційної роботи була визначена трудомісткість розробленої інформаційної системи. Крім того, було проведено розрахунок вартості роботи зі створення програми та оцінено необхідний час для її створення.

Практичне значення полягає у розробці веб-інтерфейсу, який дозволяє розробникам легко імпелментувати його у готові програмні продукти.

Актуальність даного програмного продукту визначається у недостатньо відкритому функціоналі POS систем кав’ярень та магазинів.

Список ключових слів: КОМП’ЮТЕР, КАВ’ЯРНЯ, ПРОДУКТИ, FASTAPI, PYTHON, POS.

ABSTRACT

Explanatory note: 45 p., 21 fig., 3 add., 20 sources.

Object of development: a web interface with access to interaction with the database.

The purpose of the qualification work: development of a web interface using FastAPI in the python programming language with documentation for developers.

In the introduction section of the qualification work, an analysis was carried out and the current state of the problem was considered, the specific purpose of the research and the field of its application were defined, the justification of the relevance of the chosen topic was given, and the statement of the problem was detailed.

In the first chapter of the qualification work, a detailed analysis of the subject area was conducted, the relevance of the task and development goals was considered, a specific statement of the task was formulated, and the requirements for software implementation, used technologies and software tools were determined.

In the second chapter of the qualification work, available solutions were analyzed, platforms were selected for development, design and development of the program was carried out. Detailed information about the operation of the program was also provided, including a description of the algorithm and structure of its operation. In the section, the process of calling and downloading the program was considered, input and output data were defined, and the characteristics of the parameters of the technical means were provided.

In the economic section of the qualification work, the labor intensity of the developed information system was determined. In addition, the calculation of the cost of work on the creation of the program was carried out and the time required for its creation was estimated.

The practical value is to develop a web interface that allows developers to easily implement it into finished software products.

The relevance of this software product is determined by insufficiently open functionality of POS systems in coffee shops and grocery shops.

List of keywords: COMPUTER, CAFFEE, PRODUCTS, FASTAPI, PYTHON, POS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

СУБД – система управління базами даних.

PYTHON – високорівнева мова програмування, яка переважно використовується для бекенд веб-розробки.

FastAPI – відкрита бібліотека Python для швидкої розробки захищених і комплексних веб-інтерфейсів

POS (point of sale) – програмне забезпечення, яке використовується для здійснення платежів та обробки транзакцій при продажу товарів або послуг.

API (Application Programming Interface) - це набір функцій та протоколів, які дозволяють різним програмам обмінюватися даними.

REST (Representational State Transfer) - це стандартний стиль архітектури для розробки веб-сервісів, що базується на принципі роботи з ресурсами через HTTP протокол.

UI – користувацький інтерфейс

ВСТУП

Інтернет став неодмінною складовою повсякденного життя в сучасному світі. Завдяки його доступності та широкому охопленню, люди отримали безмежний доступ до інформації, комунікації та можливостей. Інтернет перетворився на потужний інструмент, який проникає в різні сфери нашого життя, включаючи освіту, бізнес, зв'язок, розваги та багато іншого.

Доступність Інтернету зробила світ більш зв'язаним та глобальним. Незалежно від місця проживання, люди можуть спілкуватися, обмінюватися ідеями, знаходити інформацію та здійснювати операції онлайн. Інтернет дозволяє нам вести відеоконференції з далекими родичами, замовляти товари з будь-якої точки світу, вивчати нові навички через онлайн-курси та безліч інших можливостей.

API є ключовим поняттям в сфері програмування та розробки програмного забезпечення. Він відкриває двері до можливостей інтеграції і обміну даними між різними програмними системами, чим покращує наявний розвиток інтернету для благ і простоти життя людства. API визначає набір правил, протоколів та інструментів, які дозволяють розробникам створювати програми, які взаємодіють з іншими програмами, сервісами або пристроями.

Завдяки API розробники можуть використовувати готовий функціонал інших програм або сервісів без необхідності розробляти все з нуля. Вони можуть використовувати готові веб-інтерфейси для інтеграції своїх додатків з соціальними мережами, платіжними системами, картографічними сервісами та багатьма іншими ресурсами. Це дозволяє прискорити розробку, покращити функціональність та розширити можливості програмного продукту.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Загальні відомості з предметної області

FastAPI - це сучасний веб-фреймворк для створення швидких та масштабованих API на мові програмування Python. Він пропонує простий та ефективний спосіб створення веб-серверів з високою продуктивністю та підтримкою асинхронної обробки запитів. FastAPI підтримує автоматичну генерацію документації на основі типів даних, що спрощує розробку, тестування та документування API. Завдяки вбудованій підтримці стандарту OpenAPI та автоматичному перевірці типів даних, FastAPI дозволяє створювати надійні та добротні API для різних застосувань. Цей фреймворк володіє широким спектром можливостей, таких як маршрутизація, обробка запитів HTTP, аутентифікація, авторизація та багато інших.

Рис. 1.1. Логотип фреймворку FastAPI

Розвиток API є невід'ємною складовою сучасного технологічного ландшафту. API дозволяє програмним додаткам взаємодіяти між собою, обмінюючись даними та функціональністю. Оскільки технології швидко розвиваються, очікується, що API також будуть еволюціонувати в майбутньому. Перспективи розвитку API:

- Зростання популярності RESTful API: REST (Representational State Transfer) є архітектурним стилем, який використовується для побудови веб-служб. RESTful API є легкими для розуміння та використання, тому їх популярність продовжуватиме зростати.

- Захист даних і безпека: З усе більшим обсягом обміну даними через API, забезпечення безпеки стає дедалі важливішим аспектом. У майбутньому очікується зростання захисту API шляхом використання нових методів шифрування, аутентифікації та авторизації, таких як двофакторна аутентифікація, токени доступу та блокчейн-технології.

- З'єднання різних пристроїв та платформ: Зростання Інтернету речей (IoT) призводить до збільшення кількості пристроїв, які можуть взаємодіяти з API. У майбутньому API будуть спроектовані таким чином, щоб бути сумісними з різними пристроями та платформами, включаючи мобільні пристрої, розумні домашні пристрої, автомобілі та більше.

1.2 Призначення розробки та область застосування

Веб-інтерфейс який розробляється, призначений для використання розробниками по всій території України, які зацікавлені у інтегруванні його у POS застосунки кав’ярень і магазинів.

Програма надасть більше можливостей для розробників, що дозволить отримувати гостям більше інформативності, щодо кількості, складу та ціни їх улюблених десертів,напоїв або продуктів. Як показує дослідження, POS системи не мають достатнього функціоналу, для інформування гостей, щодо кількості товарів на залишку, тому не рідко бувають випадки, коли люди з сумом повертаються додому, дізнавшись, що їх улюбленого товару не має в наявності.

1.3 Підстава для розробки

Підставами для розробки та виконання кваліфікаційної роботи є:

- освітня програма 121 «Інженерія програмного забезпечення»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 350-с від 16.05.2023 р;
- завдання на кваліфікаційну роботу на тему «Розробка веб-додатка взаємодії з базою даних MySQL за допомогою мови програмування Python та фреймворку fastAPI».

1.4 Постановка завдання

Розробити веб-інтерфейс для взаємодії між окремими закритими POS системами за допомогою FastAPI, python, бази даних MySql у середовищі програмування Pycharm.

Функціональні особливості:

- додавання нових продуктів, редагування цін, складу, алергенів, типу товару і видалення старих продуктів;
- реєстрація користувачів, зміна даних користувача;
- зрозуміла і структрована документація для розробників.

1.5 Вимоги до програми або програмного виробу

1.5.1 Вимоги до функціоналих характеристик

Кінцевий продукт має дотримуватися наступних функціональними вимог:

- реєстрація: форма, що дозволяє користувачеві зареєструватися в базі, керувати особистою інформацією, редагуючи чи видаляючи її;
- зберігання і захист даних кав’ярень та магазинів у базі даних;
- додавання, видалення і редагування товарів ;
- вивід всіх назв зареєстрованих закладів.

1.5.2 Вимоги до інформаційної безпеки

Для створення безпечного функціонування програми необхідно реалізувати наступне:

- перевірка введених даних на відповідність довжині, типам та обов'язковим полям, щоб упевнитись в їх правильності;
- застосування механізмів збереження даних у базі даних, щоб забезпечити їх надійність та доступність;
- використання заходів безпеки для захисту системи від хакерських атак, таких як SQL-ін'єкція та перехоплення сесій. Це може включати реалізацію механізмів фільтрації та нормалізації вхідних даних;
- Забезпечення безпечного зберігання даних у базі даних, включаючи використання шифрування та інших методів захисту.

1.5.3 Вимоги до складу та параметрів технічних засобів

Для роботи системи необхідно використовувати серверну ЕОМ, що має щонайменше:

- 2 Гб ОЗУ;
- 4 ГБ вільного місця на жорсткому диску з системою;

1.5.4 Вимоги до інформаційної та програмної сумісності

Розробникам необхідно мати вказані середовища для доступу до розроблюваного веб-додатку:

- операційна система: Windows, Linux, Mac OS;
- будь-який браузер чи середа розробки;
- постійне підключення до мережі Інтернет.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми.

В ході виконання даної кваліфікаційної роботи була виконана розробка відкритого веб-інтерфейсу .

Призначення програми:

- захищена взаємодійя з базою даних;
- створення аккаунту;
- безпечна зміна токена чи паролю користувача;
- видалення користувача;
- додавння нових продуктів;
- редагування або видалення продуктів;
- пошук продуктів за типом.

2.2. Опис застосованих математичних методів.

При проектуванні та розробці API не використовувалися складні математичні методи або алгоритми. Розрахунки та обчислення, які використовувалися, базувалися на простих арифметичних операціях.

2.3. Опис використаної архітектури та шаблонів проектування.

При проектуванні API була використана REST архітектура.

У REST-архітектурі веб-ресурси (наприклад, дані, функції або послуги) відображаються як URI (ідентифікатори ресурсів) і можуть бути доступні для створення (POST), отримання (GET), оновлення (PUT) або видалення (DELETE) за допомогою відповідних HTTP-запитів.

Основні принципи REST-архітектури включають наступне:

- ресурси (Resources): Кожний компонент системи, який може бути доступним через мережу, вважається ресурсом. Ресурси ідентифікуються унікальними ідентифікаторами URI (Uniform Resource Identifier), такими як URL (Uniform Resource Locator);

- уніфікований інтерфейс (Uniform Interface): У REST-архітектурі використовується спільний набір обмежень на інтерфейс комунікації між клієнтом і сервером. Це включає використання HTTP методів (GET, POST, PUT, DELETE) для взаємодії з ресурсами і використання стандартних кодів статусу HTTP для передачі інформації про стан запиту;

- безстанність (Statelessness): Клієнт повинен надавати всю необхідну інформацію для виконання запиту, і сервер не повинен зберігати стан клієнта між запитами. Кожний запит вважається самодостатнім;

- кешування (Caching): Сервери можуть надавати відповіді з позначками (наприклад, заголовками HTTP), які дозволяють клієнтам кешувати ці відповіді. Це дозволяє знизити навантаження на сервер та покращити продуктивність;

- клієнт-серверна архітектура (Client-Server): REST передбачає розділення клієнта (користувача або додатку) і сервера (системи, що надає ресурси) як окремих компонентів, що можуть розвиватись незалежно один від одного.

REST-архітектура надає простий і легкозрозумілий підхід до проектування API, заснований на принципах веб-стандартів. Вона сприяє створенню гнучких, масштабованих та легко розширюваних веб-сервісів.

За шаблон проектування був взятий MVC.

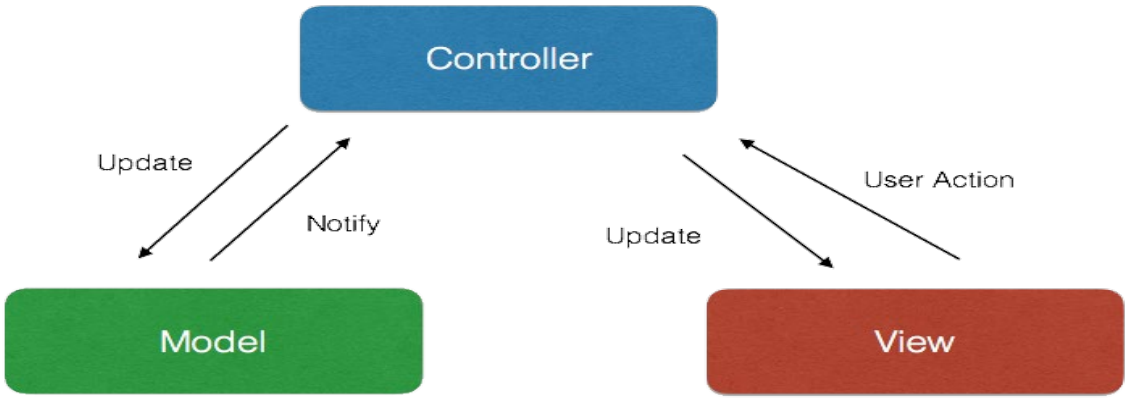

Рис 2.1. Схема принципу роботи MVC шаблону проектування.

MVC - це архітектурний патерн, який розділяє програму на три компоненти: модель (дані та бізнес-логіка), подання (відображення даних) та контролер (обробка вхідних подій та керування взаємодією між моделлю та поданням). Він полегшує розробку, підтримку та тестування програмного забезпечення, забезпечує більшу гнучкість та повторне використання коду.

2.4. Опис використаних технологій та мов програмування.

При створенні цього проекту були використані наступні мови програмування та технології:

- python 3.10
- FastAPI
- SQLAlchemy
- Uvicorn
- swagger

Python — це високорівнева, інтерпретована та універсальна динамічна мова програмування, яка зосереджується на зручності читання коду. Зазвичай він має невеликі програми порівняно з Java і C. Він був заснований у 1991 році розробником Гвідо Ван Россумом. Python входить до числа найпопулярніших і

швидкозростаючих мов у світі. Python — потужна, гнучка та проста у використанні мова. Крім того, спільнота python дуже активна. Його використовують у багатьох організаціях, оскільки він підтримує кілька парадигм програмування. Він також виконує автоматичне керування пам'яттю.

FastAPI - це сучасний веб-фреймворк для створення швидких та масштабованих API з використанням мови програмування Python. Він базується на стандарті ASGI (Asynchronous Server Gateway Interface) та використовує потужну систему типів Python для автоматичної валідації вхідних даних та генерації документації API.

SQLAlchemy - це популярна бібліотека для роботи з базами даних у мові програмування Python. Вона надає об'єктно-реляційний (ORM) підхід до взаємодії з базами даних, що дозволяє працювати з даними, використовуючи об'єктно-орієнтований код замість написання чистого SQL.

Основні переваги SQLAlchemy:

- Кросс-платформеність: SQLAlchemy підтримує різні бази даних, такі як PostgreSQL, MySQL, SQLite, Oracle та багато інших, що дозволяє легко переносити код між різними системами управління базами даних.
- ORM функціонал: SQLAlchemy надає потужний ORM-інтерфейс, який дозволяє виконувати операції з базою даних в стилі об'єктно-орієнтованого програмування. Ви можете створювати класи, що представляють таблиці бази даних, та працювати з ними як з об'єктами, що спрощує роботу з даними.
- Можливості SQL: SQLAlchemy надає можливість використання SQL-запитів у разі потреби більшої гнучкості та контролю над запитами до бази даних. Ви можете використовувати SQL-вирази та функції для складних операцій, які не підтримуються ORM-інтерфейсом.
- Транзакції та кешування: SQLAlchemy дозволяє працювати з транзакціями для забезпечення консистентності та безпеки даних.

Вона також надає можливість кешування запитів та результатів, що покращує продуктивність додатка

Uvicorn - це високопродуктивний веб-сервер, який підтримує протокол ASGI і призначений для запуску веб-додатків, розроблених з використанням асинхронного підходу у мові програмування Python, що ідеально підходить для даного проекту.

Дані зберігаються у реляційній базі даних MySQL

MySQL - це одна з найпопулярніших та надійних систем управління базами даних (СУБД). Вона забезпечує ефективне зберігання та управління великими обсягами даних, швидкі операції читання/запису, підтримку мови SQL та розширення, таких як тригери та процедури. MySQL підтримує розподілені системи, реплікацію даних для забезпечення високої доступності та шкальованості, а також має велику спільноту користувачів та багато ресурсів для навчання та підтримки. Вона використовується у багатьох веб-додатках, великих системах електронної комерції, системах управління контентом та інших проектах, де потрібно надійне та ефективне зберігання даних.

Swagger - потужний інтерфейс для документації коду програми вбудований в фреймворк FastAPI

2.5. Опис структури програми та алгоритмів її функціонування.

Для застосунку була створена ER – діаграма (рис 2.2.) в якій описана структура даних в реляційній СУБД.

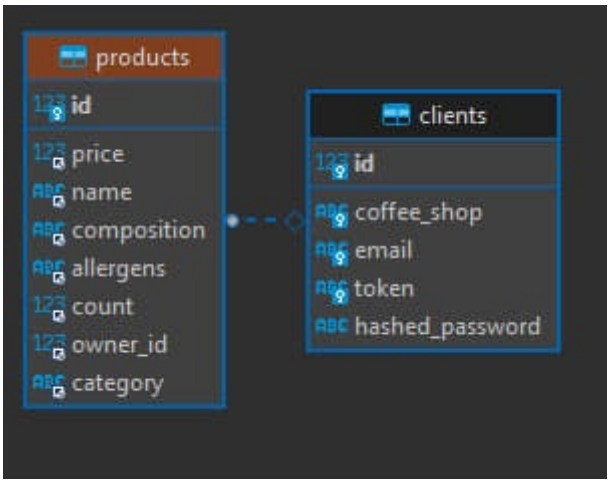

Рис 2.2. ER-діаграма

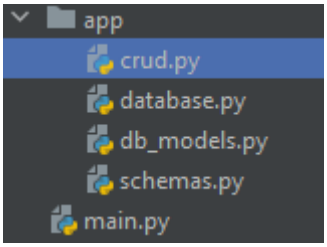

Рис 2.3. Структура програми

Файл crud.py має набір основних операцій, які використовуються для взаємодії з базами даних та збереження, отримання, оновлення та видалення даних.

Файл database.py має набір функцій для приєднання і створення сессії з базою даних.

Файл db_models.py має об’єкти СУБД, котрі створюються при першому запуску бази даних.

Файл schemas.py описує об’єкти

Файл main.py головний модуль програми в якому реалізована основна логіка REST архітектури.

2.6. Обґрунтування та організація вхідних та вихідних даних програми.

При створенні профілю користувач вводить пароль, пошту і назву закладу.

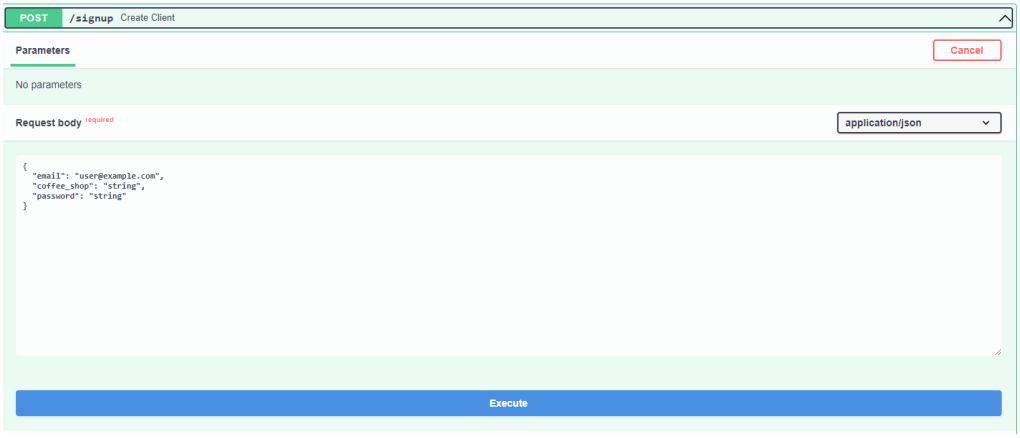

Рис 2.4. Приклад реєстрації користувча

clients | Enter a SQL expression to filter results (use Ctrl+Space)

	id	coffee_shop	email	token	hashed_password
1	1	Takava	someuser@nmu.one	2BcmH6S9W8K4ulw0kDIfyrF	$2b$12$EPEw2uF6f35BSYTIs
2	2	Pastoral	pinguin@gmail.com	7kJJKE40YmoJbYC-NQp-xX\	$2b$12$vwjmrhalS4m0EXjh7

Рис 2.5. Користувачі в базі даних

products | Enter a SQL expression to filter results (use Ctrl+Space)

	id	price	name	composition	allergens	count	owner_id
1	1	70	Круасан	борошно, масло, розп	борошно	6	2
2	2	60	Кола	вода, цукор, підсолодж	підсолоджувач	3	2

Рис 2.6. Продукти в базі даних

2.7. Опис розробленого програмного продукту.

Для початку роботи з api, необхідно перейти за його посиланням:

127.0.0.1:8000/docs#/

Рис 2.7. Посилання в браузері

Мій проект запускається через локальний сервер uvicorn.

FastAPI 0.1.0 OAS3

/openapi.json

default

POST	/signup	Create Client
POST	/revoke_token	Revoke Token
POST	/change_password	Change Password
POST	/delete_account	Delete Account
GET	/get_all_shops/	All Shops
GET	/{coffee_shop}/products/	Get Products
GET	/{coffee_shop}/products/{category}	Get Products By Category
POST	/{client_token}/products/	Create Item For Client
POST	/{client_token}/products/{product_id}	Update Product
POST	/{client_token}/delete_product/{product_id}	Delete Product

Рис 2.8. UI документація

2.7.1 Використані технічні засоби

Для забезпечення зручної роботи з сучасними браузерами, розробники рекомендують мати наступні мінімальні параметри комп'ютера:

- Центральний процесор (ЦП): Pentium 4 або еквівалент.
- Відеоадаптер: 3D адаптер від nVidia, Intel, AMD/ATI.
- Відеопам'ять: мінімум 128 МБ відеопам'яті.
- Накопичувач: щонайменше 150 ГБ вільного простору на жорсткому диску.
- Оперативна пам'ять: мінімум 2 ГБ оперативної пам'яті.

Технічні засоби, використані мною:

1. Центральний процесор (ЦП): Intel Core i5.
2. Відеоадаптер: nVidia GTX1050.
3. Відеопам'ять: 2 ГБ.
4. Накопичувач: 1 ТБ.
5. Оперативна пам'ять: 16 ГБ.
6. Клавіатура, миша, монітор.

2.7.2 Використані програмні засоби

Для розробки даного застосунку були використані наступні програмні засоби:

– PyCharm

– DBever

Як середовище для розробки вибрано PyCharm:

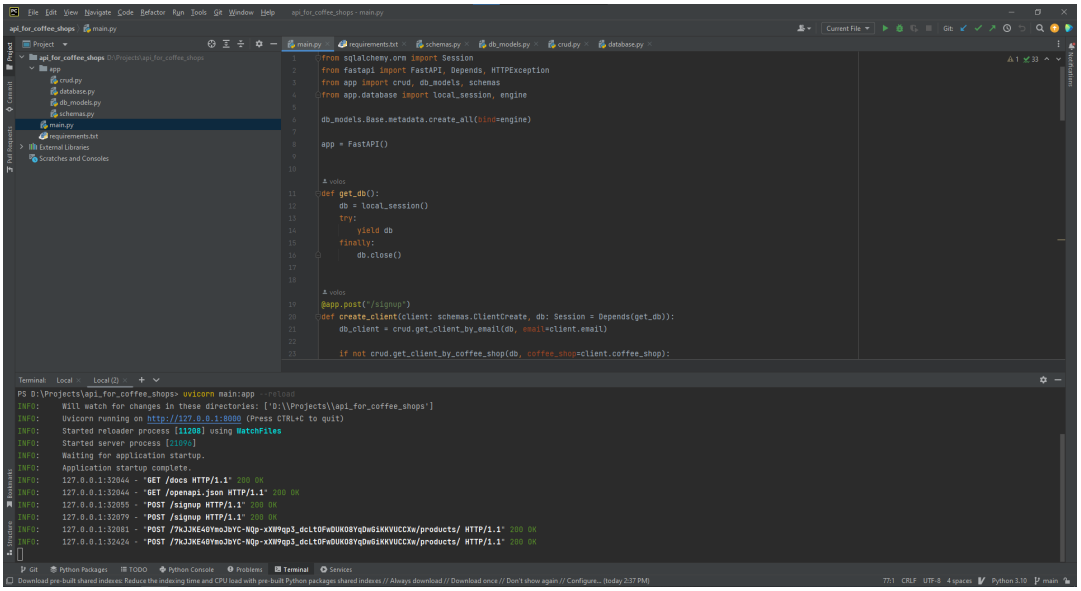

Рис 2.9. Вигляд середовища PyCharm

Застосунок для з роботи з БД було обрано DBever:

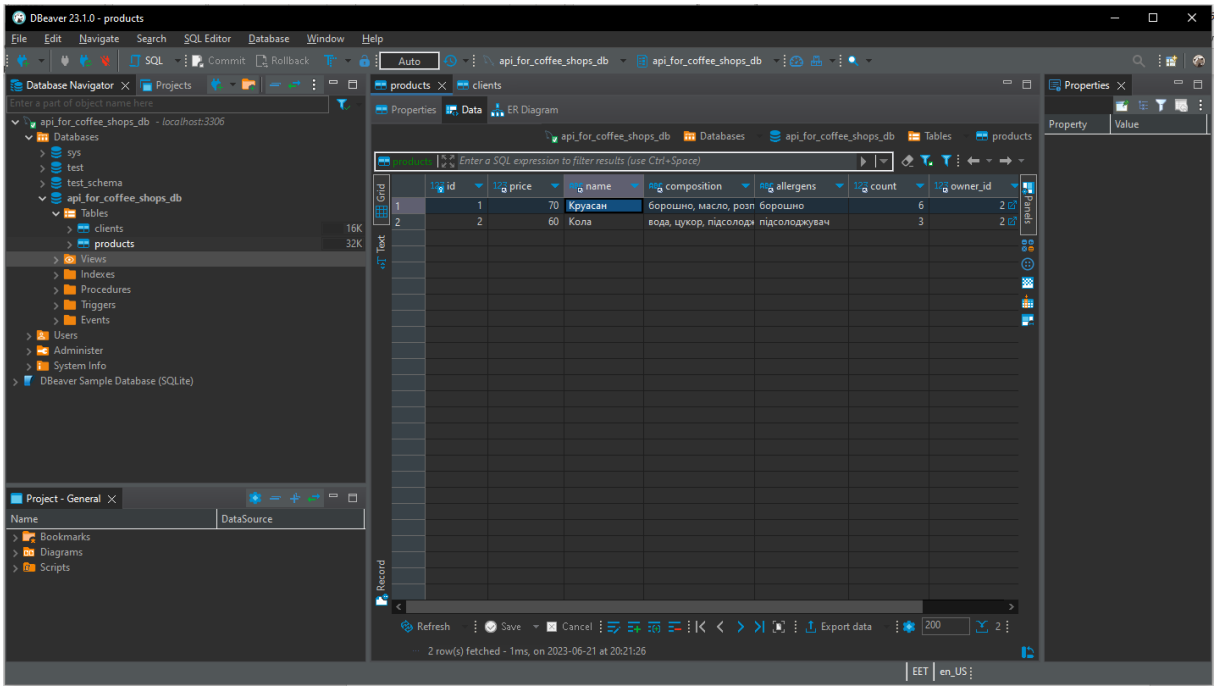

Рис 2.10. Вигляд застосунка DBever

2.7.3 Виклик та завантаження програми

Розроблений API працює на будь-якій операційній системі та в будь-якому браузері чи середовищі розробки, при правильно введеному посиланні: (URL: http://localhost:8000/docs)

2.7.4 Опис стурктурі програмного додатку

Після переходу за посиланням розробник може почати взаємодію з API як за допомогою реквестів, так і за допомогою swagger ui.

Надалі приведу приклади відсилання запитів за допомогою swagger ui.

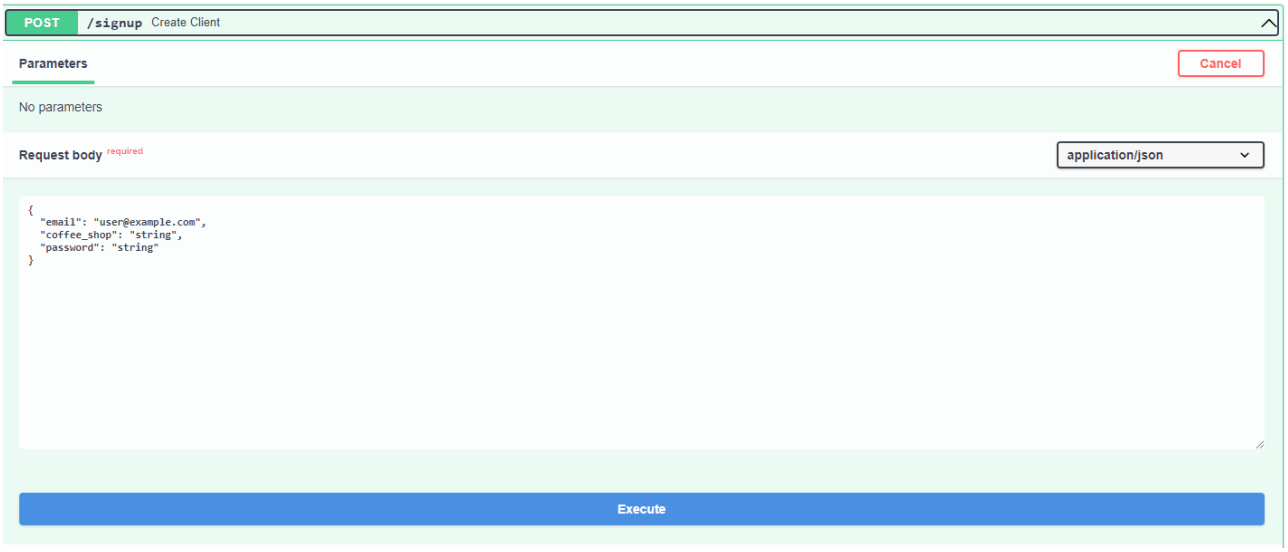

Рис 2.11. Реєстрація

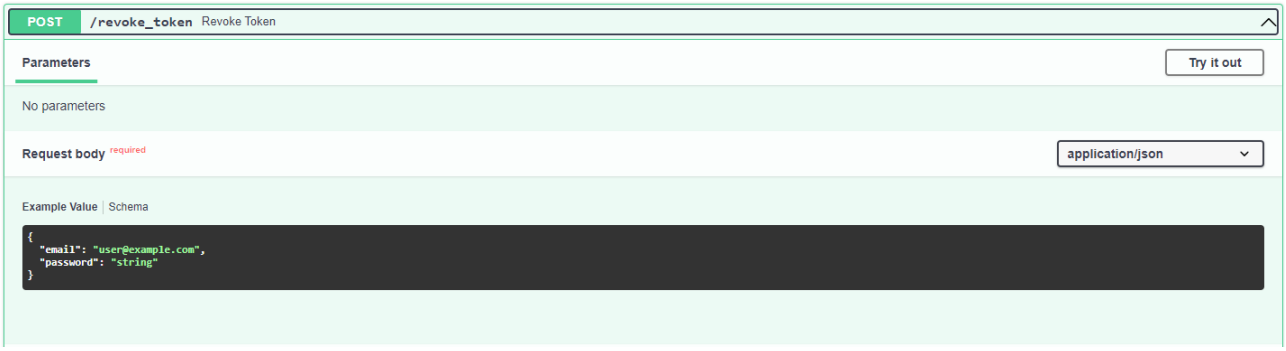

Рис 2.12. Зміна токену користувача

Рис 2.13. Зміна паролю

Рис 2.14. Видалення акаунту

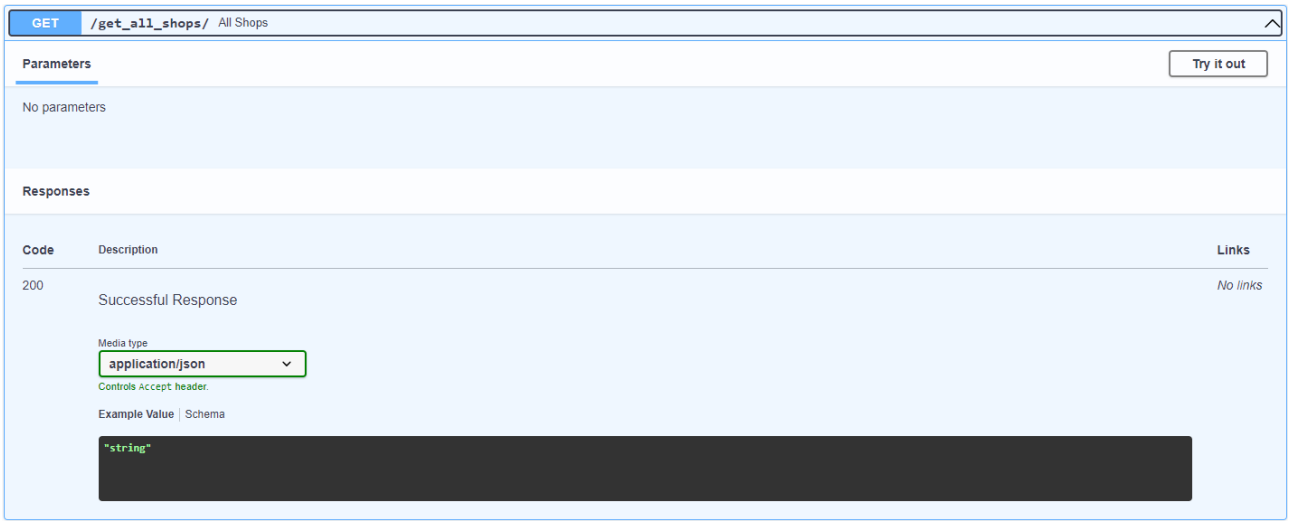

Рис 2.15. Отримання всіх назв закладів

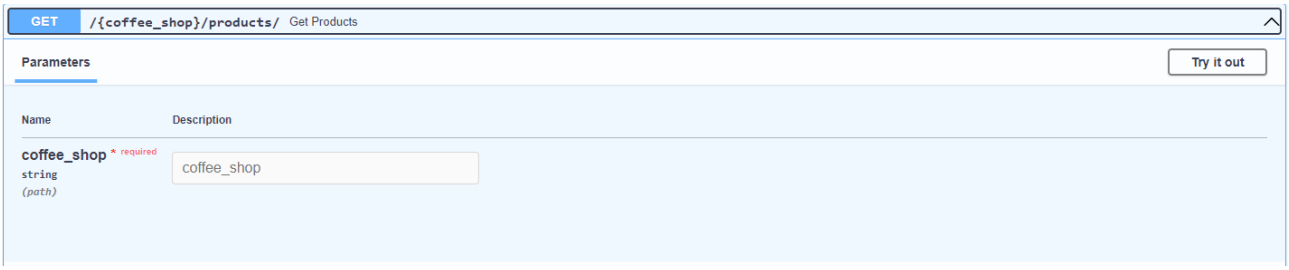

Рис 2.16. Отримання всіх продуктів у закладі

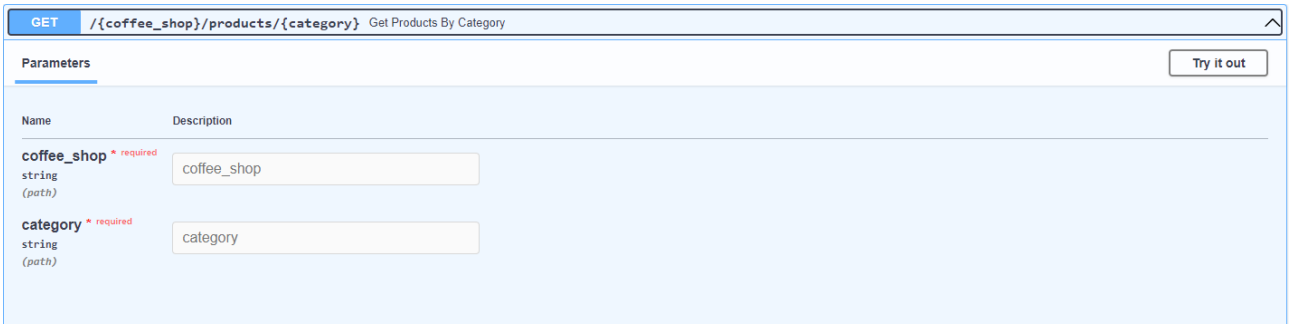

Рис 2.17. Пошук продукту за категорією

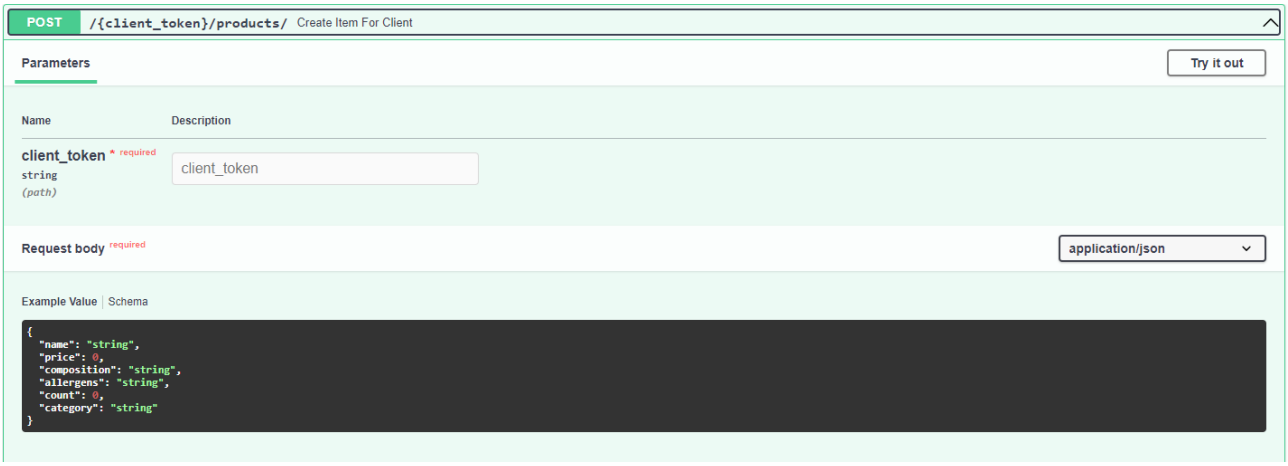

Рис 2.18.Створення продукту

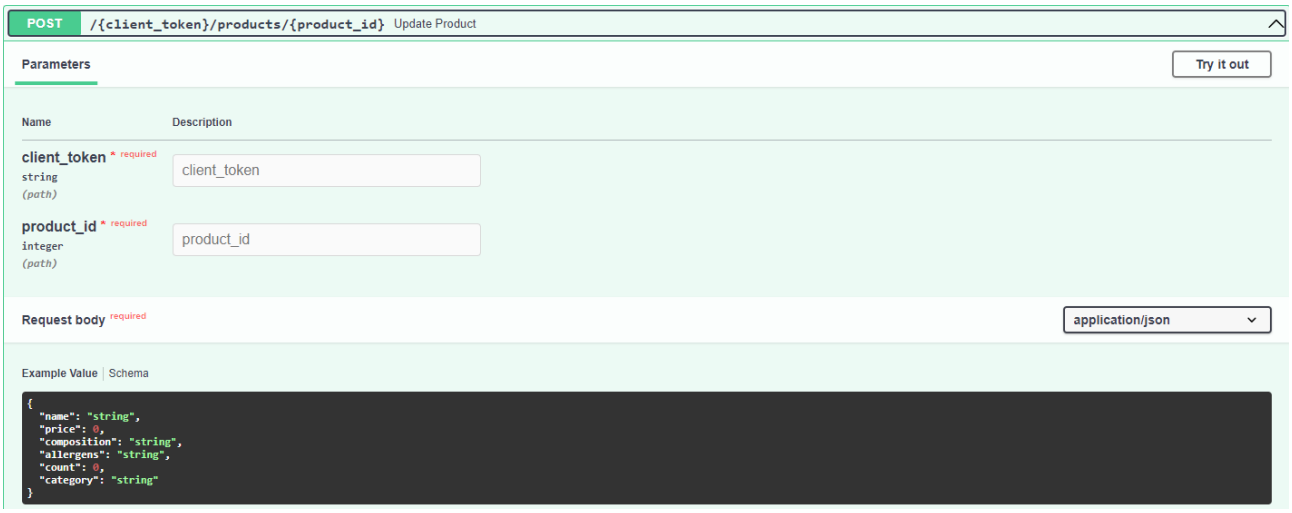

Рис 2.19. Оновлення продукту

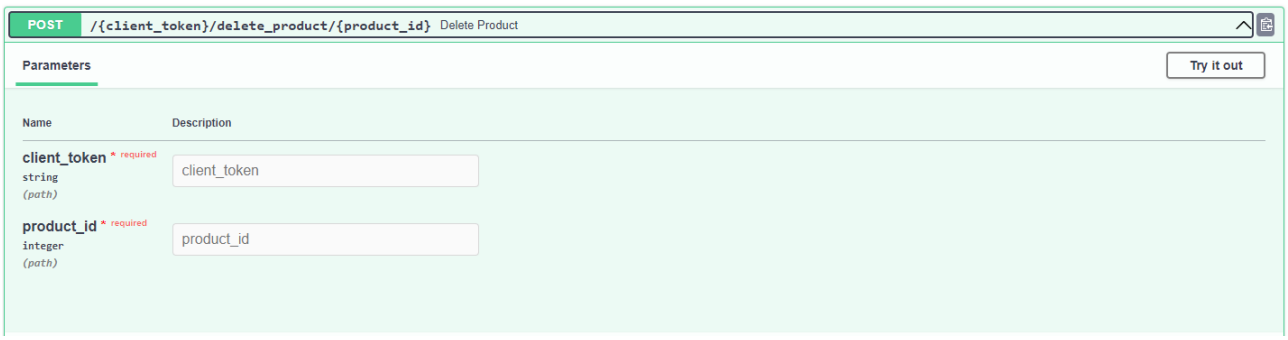

Рис 2.20. Видалення продукту

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1 Розрахунок трудомісткості та вартості розробки програмного продукту

Вихідні дані:

1. передбачуване число операторів програми – 339;

2. коефіцієнт складності програми – 1,3;

3. коефіцієнт корекції програми в ході її розробки – 0,2;

4. годинна заробітна плата розробника – 105 грн/год;

5. коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,25;

6. коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1,09;

7. вартість машино-години ЕОМ – 12 грн/год (1 грн – е/е, 6 грн – ПЗ, 5 грн - амортизація).

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки. Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,}$$

де t_o – витрати праці на підготовку й опис поставленої задачі (приймається 50);

t_u – витрати праці на дослідження алгоритму рішення задачі;

t_a – витрати праці на розробку блок-схеми алгоритму;

t_n – витрати праці на програмування по готовій блок-схемі;

$t_{отл}$ – витрати праці на налагодження програми на ЕОМ;

$t_д$ – витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у ПЗ, яке розробляється.

Умовне число операторів (підпрограм):

$$Q = q \cdot C \cdot (1 + p),$$

де q – передбачуване число операторів (339);

C – коефіцієнт складності програми (1,3);

p – коефіцієнт корекції програми в ході її розробки (0,2).

$$Q = 339 \cdot 1{,}3 \cdot (1 + 0{,}2) = 528{,}84.$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q \cdot B}{(75...85) \cdot K}, \text{ людино-годин}$$

де В – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,25);

К – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності (1,09);

$$t_u = \frac{528{,}84 \cdot 1{,}25}{85 \cdot 1{,}09} = 7{,}13 \text{ людино-годин.}$$

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25) \cdot K};$$

$$t_a = \frac{528{,}84}{20 \cdot 1{,}09} = 24{,}25 \text{ людино-годин.}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)\cdot K};$$

$$t_n = \frac{528,84}{25\cdot 1,09} = 19,4 \text{ людино-годин.}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5)\cdot K};$$

$$t_{отл} \frac{528,84}{5\cdot 1,09} = 97,03 \text{ людино-годин.}$$

– за умови комплексного налагодження завдання:

$$t_{отл}^{к} = 1,2 \cdot t_{отл};$$

$$t_{отл}^{к} = 1,2 \cdot 97,03 = 116.43 \text{ людино-годин.}$$

Витрати праці на підготовку документації:

$$t_д = t_{др} + t_{до};$$

де $t_{др}$ – трудомісткість підготовки матеріалів і рукопису;

$$t_д = \frac{Q}{(15...20)\cdot K};$$

$$t_{др} = \frac{528,84}{20\cdot 1,09} = 24,25 \text{ людино-годин.}$$

де $t_{до}$ – трудомісткість редагування, печатки й оформлення документації;

$$t_{до} = 0,75 \cdot t_{др};$$

$t_{до}$ = 0,75 · 24,25= 18,18 людино-годин.

$t_{д}$ = 24.25 + 18,18 = 42,43 людино-годин.

Отримаємо трудомісткість розробки програмного забезпечення:

t = 50 + 7,13 + 24,25 + 19,4 + 97,03 + 42,43 = 240,24 людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 240,24 людино-годин для розробки даного програмного забезпечення.

3.2 Розрахунок витрат на створення програми

Витрати на створення ПЗ $Кпо$ включають витрати на заробітну плату виконавця програми $З_{ЗП}$ витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$Кпо = З_{ЗП} + З_{МВ}, \text{ грн},$$

$З_{ЗП}$ – заробітна плата виконавців, яка визначається за формулою:

$$З_{ЗП} = t \cdot С_{ПР}, \text{ грн},$$

де t – загальна трудомісткість, людино-годин;

$С_{ПР}$ – середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата розробника становить 105 грн/год, то отримаємо:

$З_{ЗП}$ = 240,24 · 105 = 25225,2 грн

Вартість машинного часу $З_{МВ}$, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} \cdot C_M \text{, грн,}$$

де $t_{отл}$ – трудомісткість налагодження програми на ЕОМ, год;

C_M – вартість машино-години ЕОМ, грн/год.

$$З_{МВ} : 97{,}03 \cdot 12 = 1164.36 \text{ грн}$$

Звідси витрати на створення програмного продукту:

$$Кпо \quad 25225{,}2 + 1164.36 = 26389.56 \text{ грн}$$

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p} \text{, мес.}$$

де B_k – число виконавців;

F_p – місячний фонд робочого часу (при 40 годинному робочому тижні F_p = 176 годин).

Витрати на створення програмного продукту:

$$T := \frac{240{,}24}{1 \cdot 176} = 1{,}36 \text{ міс.}$$

Висновки. Час розробки даного програмного забезпечення складає 240,24 людино-годин. Таким чином, очікувана тривалість розробки складе 1,36 місяця при 40 годинному робочому тижні (місячний фонд робочого часу 176 годин), а витрати на створення програмного забезпечення складатимуть 26389.56 грн.

ВИСНОВКИ

За результатом виконання кваліфікаційної роботи було зроблено веб інтерфейс для взаємодії з POS системами закладів.

Головною метою моєї розробки, це створення зручного API. Для цього були розроблені функції, завдяки яким розробники можуть виконувати наступні дії:

- створення, редагування і видалення клієнта;
- створення, считування, редагування і видалення продуктів;
- взаємодія з API через swagger ui.

Під час виконання цих завдань було створено веб-інтерфейс, що відповідає вимогам предметної області і надає необхідну функціональність для вирішення поставлених перед проектом завдань.

Проект реалізовано за допомогою інструментів, що з легкістю дозволяють масштабувати та підтримувати його.

В «Економічному розділі» було здійснено розрахунок людино-годин, які потрібні для повної розробки інформаційної системи (176 людино/годин). Також було визначено очікувані витрати на створення проекту (26389.56 грн) і ймовірна очікувана тривалість розробки(1,36 місяці).

В результаті виконання даної кваліфікаційної роботи були досягнуті всі початково поставлені цілі. Крім того, всі вимоги, визначені під час проектування інформаційної системи, були успішно виконані.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Paul DuBois. MySQL Cookbook. — O'Reilly Media, 2014. — 866 с.
2. Luciano Ramalho. Fluent Python: Clear, Concise, and Effective Programming. — O'Reilly Media, 2015. — 790 с.
3. Zed Shaw. Learn Python the Hard Way: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code (Zed Shaw's Hard Way Series). — Addison-Wesley Professional, 2013. — 320 с.
4. Eric Matthes. Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming. — No Starch Press, 2019. — 544 с.
5. David Beazley, Brian Jones. Python Cookbook, Third Edition. — O'Reilly Media, 2013. — 704 с.
6. Ilya Grigorik. High Performance Browser Networking: What every web developer should know about networking and web performance. — O'Reilly Media, 2013. — 398 с.
7. Michael J. Hernandez. Database Design for Mere Mortals: A Hands-On Guide to Relational Database Design First Edition. — Addison-Wesley, 1996. — 440 с.
8. Stephane Faroult, Peter Robson. The Art of SQL 1st Edition. — O'Reilly Media, 2006. — 370 с.
9. Sunil Kapil. Clean Python: Elegant Coding in Python. — Apress, 2019. — 286 с.
10. Charles Bell, Mats Kindahl, Lars Thalmann. MySQL High Availability: Tools for Building Robust Data Centers. — O'Reilly Media, 2014. — 746 с.
11. MySQL. / URL: https://www.mysql.com
12. GitHub. / URL: https://github.com/
13. Uvicorn. / URL: https://www.uvicorn.org/
14. FastAPI. / URL: https://fastapi.tiangolo.com/
15. Python. / URL: https://www.python.org/
16. SQL Alchemy. / URL: www.sqlalchemy.org
17. Quick start guide. / URL: https://www.jetbrains.com/help/pycharm/quick-start-guide.html
18. Pydantic. / URL: https://docs.pydantic.dev/latest/

19. Swagger. / URL: https://swagger.io/tools/swagger-ui/
20. MartinFowler. / URL: https://martinfowler.com/

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

Даний програмний код було розміщено на GitHub за посиланням:

https://github.com/Aliencake/api_for_coffee_shops

requirements.txt // необхідні модулі для роботи програми

```
email-validator==2.0.0.post2
fastapi==0.97.0
uvicorn==0.22.0
pydantic==1.10.9
SQLAlchemy==2.0.16
bcrypt==4.0.1
```

main.py // головний модуль програми

```
from sqlalchemy.orm import Session
from fastapi import FastAPI, Depends, HTTPException
from app import crud, db_models, schemas
from app.database import local_session, engine

db_models.Base.metadata.create_all(bind=engine)

app = FastAPI()


def get_db():
    db = local_session()
    try:
        yield db
    finally:
        db.close()


@app.post("/signup")
def create_client(client: schemas.ClientCreate, db: Session = Depends(get_db)):
    db_client = crud.get_client_by_email(db, email=client.email)
```

```
        if not crud.get_client_by_coffee_shop(db, coffee_shop=client.coffee_shop):
            pass
        else:
            raise HTTPException(status_code=400, detail="Назва кав'ярні вже існує")

        if 7 > int(len(client.password)):
            raise HTTPException(status_code=400, detail="Пароль має містити більше 6 символів")
        if db_client:
            raise HTTPException(status_code=400, detail="Ця пошта вже зареєстрована")
        token = crud.create_client(db=db, client=client, coffee_shop=client.coffee_shop)
        return {"Ви успішно зареєстровані, ваш токен": token}
    # Dependency


    @app.post("/revoke_token")
    def revoke_token(client: schemas.ClientLogIn, db: Session = Depends(get_db)):
        db_client: db_models.Client = crud.get_client_by_email(db, email=client.email)
        if db_client:
            response = crud.revoke_token(db=db, db_client=db_client, input_password=client.password)
            if response:
                return response
            else:
                raise HTTPException(status_code=400, detail="Не правильний пароль")
        else:
            raise HTTPException(status_code=400, detail="Користувача не знайдено")


    @app.post("/change_password")
    def change_password(client: schemas.ClientChangePassword, db: Session = Depends(get_db)):
        if 7 > int(len(client.new_password)):
            raise HTTPException(status_code=400, detail="Пароль має містити більше 6 символів")
        db_client: db_models.Client = crud.get_client_by_email(db, email=client.email)
```

```
    if db_client:
        response = crud.change_password(db=db, db_client=db_client, client=client)
        if response:
            return response
        else:
            raise HTTPException(status_code=400, detail="Немає сумісності токену з поштою")
    else:
        raise HTTPException(status_code=400, detail="Користувача не знайдено")


@app.post("/delete_account")
def delete_account(client: schemas.ClientDelete, db: Session = Depends(get_db)):
    db_client: db_models.Client = crud.get_client_by_email(db, email=client.email)
    if db_client:
        return crud.delete_client(db=db, db_client=db_client, client=client)
    else:
        raise HTTPException(status_code=400, detail="Користувача не знайдено")

@app.get("/get_all_shops/")
def all_shops(db: Session = Depends(get_db)):
    clients = crud.get_clients(db)
    return {client.coffee_shop for client in clients}


@app.get("/{coffee_shop}/products/")
def get_products(coffee_shop: str, db: Session = Depends(get_db)):
    client: schemas.Client = crud.get_client_by_coffee_shop(db=db, coffee_shop=coffee_shop)
    if not client:
        raise HTTPException(status_code=400, detail="Кав'ярню не знайдено")
    products = crud.get_products(db, client.id)
    return products


@app.get("/{coffee_shop}/products/{category}")
def get_products_by_category(coffee_shop: str, category: str, db: Session = Depends(get_db)):
    client: schemas.Client = crud.get_client_by_coffee_shop(db=db, coffee_shop=coffee_shop)
```

```
    if not client:
        raise HTTPException(status_code=400, detail="Кав'ярню не знайдено")
    products = crud.get_products_by_category(db, client.id, category)
    return products


@app.post("/{client_token}/products/", response_model=schemas.Product)
def create_item_for_client(client_token: str, item: schemas.ProductCreate, db: Session = Depends(get_db)):
    client_id = crud.get_client_id_by_token(db=db, token=client_token)
    return crud.create_product(db=db, item=item, client_id=client_id)


@app.post("/{client_token}/products/{product_id}")
def update_product(client_token: str, product_id: int, item: schemas.ProductCreate, db: Session = Depends(get_db)):
    client_id = crud.get_client_id_by_token(db=db, token=client_token)
    response = crud.update_product(db=db, product_id=product_id, client_id=client_id, item=item)
    if response:
        return response
    else:
        raise HTTPException(status_code=400, detail="Це не ваш продукт")


@app.post("/{client_token}/delete_product/{product_id}")
def delete_product(client_token: str, product_id: int, db: Session = Depends(get_db)):
    return crud.delete_product(db=db, product_id=product_id, token=client_token)
```

crud.py

```
from bcrypt import hashpw, gensalt, checkpw
from sqlalchemy.orm import Session
from app import db_models, schemas
from secrets import token_urlsafe
from fastapi import HTTPException
```

```
def revoke_token(db: Session, db_client: db_models.Client, input_password: str):
    if checkpw(input_password.encode('utf-8'), db_client.hashed_password.encode('utf-8')):
        token = token_urlsafe(40)
        setattr(db_client, 'token', token)
        db.commit()
        db.refresh(db_client)
        return {"Ваш новий токен": token}
    else:
        return False


def change_password(db: Session, client: schemas.ClientChangePassword, db_client: db_models.Client):
    if client.token == db_client.token:
        hashed_password = hashpw(client.new_password.encode('utf-8'), gensalt())
        setattr(db_client, 'hashed_password', hashed_password)
        db.add(db_client)
        db.commit()
        db.refresh(db_client)
        return f"Ваш пароль змінено на: {client.new_password}"
    return False


def delete_client(db: Session, client: schemas.ClientDelete, db_client: db_models.Client):
    if not checkpw(client.password.encode('utf-8'), db_client.hashed_password.encode('utf-8')):
        raise HTTPException(status_code=400, detail="Не вірний пароль")
    if not client.token == db_client.token:
        raise HTTPException(status_code=400, detail="Не вірний токен")
    products: db_models.Product | [] = db.query(db_models.Product).filter(db_models.Product.owner_id == db_client.id).all()
    for product in products:
        db.delete(product)
    db.delete(db_client)
    db.commit()
    return f"Кав'ярню {db_client.coffee_shop} успішно видалено"
```

```
def get_client(db: Session, user_id: int):
    return db.query(db_models.Client).filter(db_models.Client.id == user_id).first()


def get_client_by_email(db: Session, email: str):
    return db.query(db_models.Client).filter(db_models.Client.email == email).first()


def get_client_by_coffee_shop(db: Session, coffee_shop: str):
    return db.query(db_models.Client).filter(db_models.Client.coffee_shop == coffee_shop).first()


def get_clients(db: Session):
    return db.query(db_models.Client).all()


def create_client(db: Session, client: schemas.ClientCreate, coffee_shop: str):
    hashed_password = hashpw(client.password.encode('utf-8'), gensalt())
    token = token_urlsafe(40)
    db_client = db_models.Client(email=client.email, hashed_password=hashed_password, token=token,
                                 coffee_shop=coffee_shop)
    db.add(db_client)
    db.commit()
    db.refresh(db_client)
    return token


def get_client_id_by_token(db: Session, token: str):
    return db.query(db_models.Client.id).filter(db_models.Client.token == token).first()[0]


def get_products(db: Session, client_id: int):
    return db.query(db_models.Product).filter(db_models.Product.owner_id == client_id).all()
```

```
def get_products_by_category(db: Session, client_id: int, category: str):
    return db.query(db_models.Product).filter(db_models.Product.owner_id == client_id).\
        filter(db_models.Product.category == category).all()


def update_product(db: Session, product_id: int, client_id: int, item: schemas.ProductCreate):
    db_product = db.query(db_models.Product).filter(db_models.Product.id == product_id).first()
    if not db_product:
        return False
    elif not db_product.owner_id == client_id:
        return False
    for key, value in item:
        setattr(db_product, key, value)
    db.add(db_product)
    db.commit()
    db.refresh(db_product)
    return db_product


def create_product(db: Session, item: schemas.ProductCreate, client_id: int):
    db_product = db_models.Product(**item.dict(), owner_id=client_id)
    db.add(db_product)
    db.commit()
    db.refresh(db_product)
    return db_product


def delete_product(db: Session, product_id: int, token: str):

    client_id: int = get_client_id_by_token(db, token)
    db_product: db_models.Product | None = db.query(db_models.Product)\
        .filter(db_models.Product.owner_id == client_id).filter(db_models.Product.id == product_id).first()
```

```
        db.delete(db_product)
        db.commit()
        return f"Продукт №{db_product.id} успішно видалено"
```

database.py

```
    from sqlalchemy import create_engine
    from sqlalchemy.ext.declarative import declarative_base
    from sqlalchemy.orm import sessionmaker

    SQLALCHEMY_DATABASE_URL =
"mysql+mysqlconnector://root:root@localhost:3306/api_for_coffee_shops_db"

    engine = create_engine(
        SQLALCHEMY_DATABASE_URL
    )
    local_session = sessionmaker(autocommit=False, autoflush=False, bind=engine)


Base = declarative_base()
```

db_models.py

```
    from sqlalchemy import Column, ForeignKey, Integer, String
    from sqlalchemy.orm import relationship

    from .database import Base


    class Client(Base):
        __tablename__ = "clients"

        id = Column(Integer, primary_key=True, index=True)
        coffee_shop = Column(String(255), unique=True, index=True)
        email = Column(String(255), unique=True, index=True)
        token = Column(String(255), unique=True, index=True)
        hashed_password = Column(String(255))

        products = relationship("Product", back_populates="owner")
```

```
class Product(Base):
    __tablename__ = "products"

    id = Column(Integer, primary_key=True, index=True)

    name = Column(String(255), index=True)
    price = Column(Integer, index=True)
    composition = Column(String(255), index=True)
    allergens = Column(String(255), index=True)
    count = Column(Integer, index=True)
    owner_id = Column(Integer, ForeignKey("clients.id"))
    category = Column(String(255), index=True)

owner = relationship("Client", back_populates="products")
```

schemas.py

```
from pydantic import BaseModel, EmailStr


class ProductBase(BaseModel):
    name: str
    price: int
    composition: str | None = None
    allergens: str | None = None
    count: int = 0
    category: str | None = None


class ProductCreate(ProductBase):
    pass


class Product(ProductBase):
    id: int
    owner_id: int
```

```
    class Config:
        orm_mode = True


class ClientBase(BaseModel):
    email: EmailStr
    coffee_shop: str


class ClientCreate(ClientBase):
    password: str


class ClientLogIn(BaseModel):
    email: EmailStr
    password: str


class ClientChangePassword(BaseModel):
    token: str
    new_password: str
    email: EmailStr


class ClientDelete(BaseModel):
    password: str
    token: str
    email: EmailStr


class Client(ClientBase):
    id: int
    products: list[Product] = []

    class Config:
```

```
orm_mode = True
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ДОКУМЕНТІВ НА ОПТИЧНОМУ НОСІЇ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_ Волошко.docx	Пояснювальна записка до дипломного проекту. Документ Word.
Диплом_ Волошко.pdf	Пояснювальна записка до дипломного проекту в форматі PDF
Програма	
Program.rar	Архів. Містить коди програми і відкомпільовану програму
Презентація	
Презентація_Волошко.pptx	Презентація дипломного проекту