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 *Бізюкова Владислава Євгенійовича*
(ПІБ)

академічної групи *121-17-1*
(шифр)

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

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

на тему: *Розробка веб-додатка товарно-інформаційної системи на мові Python та фреймворку Django*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*доц. Сироткіна О.І.*			
розділів:				
спеціальний	*доц. Сироткіна О.І.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2021

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

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

_____ І.М. Удовик
(підпис) (прізвище, ініціали)

« » _____ 2021 року

ЗАВДАННЯ
на кваліфікаційну роботу

бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *121-17-1* *Бізюкова Владислава Євгенійовича*
(група) (прізвище та ініціали)

тема кваліфікаційної роботи *Розробка веб-додатка товарно-інформаційної системи на мові Python та фреймворку Django*

затверджена наказом ректора НТУ «ДП» від *07.06.2021р.* № *317-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів виробничої практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі.*** ***Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2021 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2021 р.***

Завдання видав _____ *доц. Сироткіна О.І.*
(підпис) (посада, прізвище, ініціали)

Завдання прийняв до виконання _____ *Бізюков В.Є.*
(підпис) (прізвище, ініціали)

Дата видачі завдання: *14.01.2021 р.*

Термін подання кваліфікаційної роботи до ЕК: *11.06.2021 р.*

РЕФЕРАТ

Пояснювальна записка: n с., n рис., n табл., n дод., n джерел.

Об’єкт розробки: веб-додаток товарно-інформаційної системи.

Мета кваліфікаційної роботи: створення зручного застосунку для реалізації торговельних відносин між клієнтом та фермером-продавцем вертикальної ферми, де клієнт може переглядати список товарів (овочів, ягід та ін.), оформляти замовлення, переглядати історію замовлень, сплачувати покупки та обирати спосіб доставлення. Фермер має здатність додавати/оновлювати інформацію о фермах, складах та товарах, котрі він зрощує. Також він має здатність продавати товар, переглядати історію продаж та увесь функціонал клієнта. Обидві сторони можуть поповнювати свій баланс у додатку, проте виводити його може тільки фермер.

У вступі проведено огляд предметної галузі та описано сучасний стан проблеми, визначено мету кваліфікаційної роботи й галузь її застосування, обґрунтовано актуальність обраної теми та сформульовано формулювання завдання.

У першому розділі проведено аналіз предметної галузі, виконано дослідження наявних аналогів та визначено їх недоліки, сформульовано причину необхідності розробки, призначення розробки, доведено її актуальність, зазначені вимоги до реалізації програмного забезпечення, технологій та програмних засобів.

У другому розділі сформульовано функціональне та експлуатаційне призначення програми, обрано інструментарій для розробки та визначено особливості процесу конструювання програмного забезпечення, виконано проєктування та розробку програми, визначено вхідні та вихідні дані, описаний процес завантаження, вигляд інтерфейсу користувача та основні принципи функціонування програми.

В економічному розділі був проведений розрахунок трудомісткості розробленої інформаційної системи, було підраховано вартість роботи зі створення програми та визначено час на його створення.

Практичне значення полягає у створенні застосунку, що реалізує зручний користувацький інтерфейс, який надає можливість обирати товар, робити замовлення, сплачувати його та обирати метод доставлення/продавати вирощений товар.

Актуальність веб-додатку визначається відсутністю конкурентів на ринку та можливістю робити попередні замовлення онлайн, а не йти до супермаркету, а також можливістю конкурентоспроможності перед великими магазинами та мережами супермаркетів.

Список ключових слів: ВЕБ-ДОДАТОК, ВЕБСАЙТ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, ІНФОРМАЦІЙНА СИСТЕМА, ФЕРМА, ВЕРТИКАЛЬНА ФЕРМА, DJANGO.

ABSTRACT

Explanatory note: n p., n figs., n tabl., n apps., n sources.

Object of development: web application of the information system of the trade application

The purpose of the qualification work: to create a convenient application for the implementation of trade relations between the client and the farmer-seller of the vertical farm, where the client can view the list of goods (vegetables, berries, etc.), place an order, view order history, pay purchases and choose delivery. The farmer has the ability to add / update information about the farms, storages and goods he grows. He also has the ability to sell goods, view sales history and all customer functionality. Both parties can replenish their balance in the application, but only the farmer can withdraw it.

The introduction provides an overview of the subject area and describes the current state of the problem, defines the purpose of the qualification work and the field of its application, substantiates the relevance of the chosen topic and formulates the problem.

In the first section the analysis of the subject area is carried out, research of existing analogues is carried out and their shortcomings are defined, the reason of necessity of development is formulated, purpose of development is proved, its urgency is proved, requirements to realization of the software, technologies and software are specified.

The second section formulates the functional and operational purpose of the program, selects tools for development and identifies features of the software design process, design and development of the program, defines input and output data, describes the download process, user interface and basic principles of the program.

In the economic section, the complexity of the developed information system was calculated, the cost of work on creating the program was calculated and the time for its creation was determined.

The practical significance lies in the creation of an application that implements a user-friendly interface that allows you to choose the product, place an order, pay for it and choose the method of delivery / sale of the grown product.

The relevance of the web client is determined by the lack of competitors in the market, as well as the possibility of competitiveness in front of large stores and supermarket chains.

Keyword list: WEB CLIENT, WEBSITE, SOFTWARE, INFORMATION SYSTEM, FARM, VERTICAL FARM, DJANGO.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ІС – інформаційна система;

ЕОМ – електронно-обчислювальна машина;

ПЗ – програмне забезпечення;

HTTP – HyperText Transfer Protocol;

DOM – Document Object Model;

HTML – HyperText Markup Language;

CSS – Cascading Style Sheets;

JSON – JavaScript Object Notation;

UX/UI – User Interface/User Experience.

ВСТУП

У нашому середовищі на цей момент вартість овочів сильно коливається від неконтрольованих факторів та пори року, а також у цю вартість у більшій мері входить комісії та інше. Завдяки нашому веб-додатку можна буде замовляти собі різні культури у будь-який час за меншу вартість.

Галуззю застосування об’єкта розробки є фермерство.

Тематика даної кваліфікаційної роботи присвячена розробці веб-додатка для торговельно-інформаційної системи

Сучасній людині у нашій країні для покупки плодово-овочеві культури потрібно йти до супермаркету, де товари можуть бути не дуже свіжі, або йти до овочевих ринків, де ціна значно вище. Основною ідеєю веб-додатку є спроможність людини планувати свої потреби на наступний проміжок часу та замовляти необхідну їй кількість овочів, ягід та ін. за значно нижчу ціну або купувати з певною націнкою на певний час, проте ця ціна все одно буде нижче ніж у магазинах. Найважливішою перевагою є те, що товари будуть свіжими та купівля/доставлення до дому буде комфортнішою.

Також у цьому веб-додатку можна буде зареєструватися у якості фермера, котрий у себе може установити вертикальну ферму та продавати урожай, вирощений вдома чи у будь-якому придатному для вирощування місці.

Метою даної кваліфікаційної роботи є створення зручного користувацького інтерфейсу інформаційної системи для покупки/продажу рослинних культур за допомогою програмного забезпечення.

Для реалізації поставленої мети необхідно виконати наступні завдання:

- проаналізувати предметну галузь;
- дослідити наявні аналоги та виділити їх недоліки;
- визначити основні вимоги до програмного продукту;
- обрати архітектуру системи, шаблони проєктування та методологію життєвого циклу розробки;

- проаналізувати варіанти та вирішити які інструменти використовувати для розробки;
- розробити веб-додаток торговельно-інформаційної системи.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Загальні відомості з предметної галузі

Людині потрібні харчові продукти, основною масою яких є овочі та фрукти. Зараз основним джерелом постачання плодово-овочевої продукції до кінцевих споживачів є агропромислові підприємства, які вирощують продукцію застосовуючи агротехнології або приватні господарства, продукція яких є екологічнішою, але значно дорожче. Основними шляхами постачання такої продукції є роздрібна мережа, ринок сільгосппродукції та власне, присадибні ділянки людей.

На 2018 рік середнє споживання овочів і ягід на рік в Україні становило між 150 та 250 кг на людину. За середню величину можна взяти 200 кг/рік, тобто трохи більше ніж 500 Гр на день. Якщо взяти за основу сім'ю з двох людей, то їм потрібно 1 кг на день. Якщо вони проаналізують їх потреби, то вони зможуть замовляти собі на кожен тиждень по 7 кг овочів та економити до 20 грн за кожен кілограм. Так само немає потреби у тривалому зберіганні овочів та коренеплодів, що є значно зручнішим ніж попередні закупівлі великого обсягу плодово-овочевої продукції з подальшим зберіганням у застосованих до цього приміщеннях, тобто використання застосунку зекономить гроші та час та надасть людині споживати завжди свіжу продукцію.

Основною проблемою для споживача є або завелика ціна, або свіжість товару. Для тих, хто вирощує щось вдома або на городі є проблема з надлишковим врожаєм.

У цьому проєкті буде реалізована система з надаванням фермерам вертикальних ферм для вирощування врожаю на протягом цілого року та можливістю віддавати частку культур на реалізацію, клієнти ж своєю чергою зможуть замовляти потрібну собі кількість овочів на певний проміжок часу. Наприклад, фермер вирощує огірки, а певному клієнту через 3 тижні потрібно

десь взяти 4 кг огірків. У веб-додатку фермер зможе подати заявку на продаж певної кількості огірків, котрі будуть стиглі до певного часу, а клієнт зможе зробити заявку на купівлю 4 кг огірків з усієї кількості до того часу.

Попри те, що ідея доволі проста, прямих конкурентів на ринку немає.

1.2. Призначення розробки та галузь застосування

Під час виконання кваліфікаційної роботи було поставлено завдання розробити програмний продукт «Веб-додаток торговельно-інформаційної системи».

Причиною виникнення необхідності розробки програмного забезпечення являється відсутність аналогів в країні, що забезпечують надають можливість замовляти овочі та ягоди на майбутній час з економією часу та фінансів.

Галузь застосування даного продукту – фермерство, а саме – ринок торговельно-інформаційних систем.

Призначення даної розробки – це надання зручного та інтуїтивно зрозумілого інтерфейсу для взаємодії з інформаційною системою для наступних осіб:

- Клієнта;
- Фермера;
- Адміністратора.

1.3. Підстави для розробки

В кінці навчання студент виконує кваліфікаційну роботу. Тема роботи узгоджується з керівником кваліфікаційної роботи, випускаючою кафедрою.

Підставою для розробки кваліфікаційної роботи на тему «Розробка веб-додатку торговельно-інформаційної системи» є наказ по Національному технічному університету «Дніпровська політехніка» 317-с від 07.06.2021р.

1.4. Постановка завдання

Метою кваліфікаційної роботи є створення зручного веб-додатку для купівлі/продажу продовольчих товарів, а саме овочів, ягід та ін.

Призначення програмного продукту – забезпечити зручний інтерфейс, що надає можливість клієнту переглядати каталог товарів, робити замовлення на майбутній час чи купляти одразу, а також для фермера додавати/оновлювати інформацію про культури, котрі він зростив.

Веб-додаток є складовою приватного проєкту по збільшенню популярності вертикальних ферм, а також для економії пересічних споживачів і заробітку фермерів.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Кінцевий продукт повинен дотримуватися наступних функціональних вимог:

- можливість легкого підключення клієнтської частини;
- дані повинні оброблятися та зберігатися до бази даних;
- дані повинні братися з бази даних, оброблятися та надсилатися до клієнта.

1.5.2. Вимоги до інформаційної безпеки

Реєстрація користувачів реалізується через заповнення своїх даних (логін, пароль та адрес електронної пошти) та токену.

Також клієнт повинен отримувати лише ті дані, котрі йому потрібні та не повинен отримувати технічну інформацію.

1.5.3. Вимоги до складу та параметрів технічних засобів

Для підтримки веб-додатку, слід дотримуватися таким технічним вимогам:

- наявність доступу до мережі інтернет;
- наявність клієнтської частини додатку;
- пристрої введення інформації;
- пристрої виведення інформації.

1.5.4. Вимоги до інформаційної та програмної сумісності

Інформаційна система потребує від користувача мати середовище з такими складовими:

одна з операційних систем: Windows, Linux, Mac OS, Android, IOS;

для користування на десктопній системі – Google Chrome, для Android чи IOS - мобільний додаток Google Chrome/Safari;

постійний доступ до мережі Інтернет.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ

2.1. Функціональне призначення програми

Розроблений програмний продукт призначений для використання двох типів користувачів (клієнтів та фермерів). У кожного з них є свої функціональні можливості.

Клієнт може:

- Переглядати список товарів;
- Додавати товари до кошика;
- Робити попереднє замовлення на товар;
- Купувати наявний товар;
- Сплачувати свої покупки;
- Обирати варіант доставлення товару;
- Редагувати свої персональні дані;
- Поповнювати свій рахунок.

Фермер на додачу функцій клієнта також може:

- Додавати новий склад;
- Додавати новий товар;
- Редагувати товар;
- Видаляти товар;
- Продавати свій товар;
- Виводити гроші зі свого балансу.

Ще для адміністрування системи потрібен обліковий запис суперкористувача з правами адміністратор, котрий має такі можливості:

- Переглядати список товарів;
- Додавати товар;

- Видаляти товар;
- Створювати облікові записи;
- Блокувати угоди;
- Видаляти угоди;
- Блокувати заявки;
- Видаляти заявки;
- Активувати облікові записи;
- Деактивувати облікові записи;
- Додати службу доставлення;
- Додати платіжну систему;
- Перевіряти статус угоди.

На рис. 2.1 зображена діаграма використання для основних учасників системи.

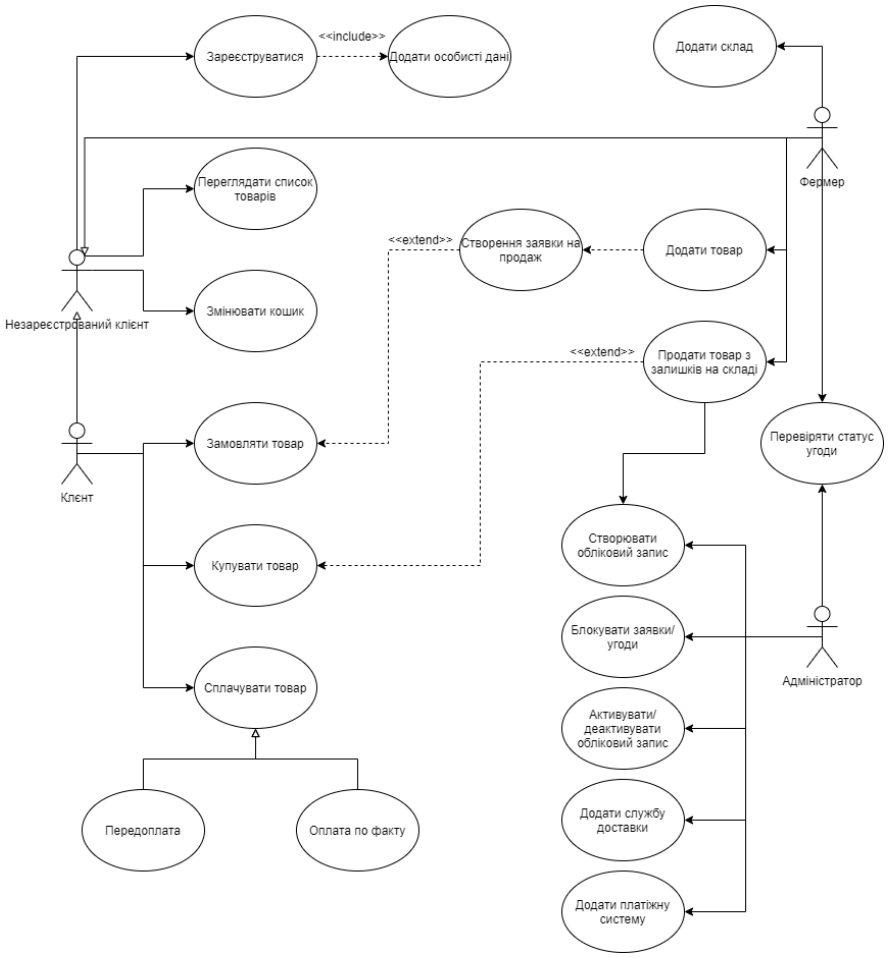

Рис. 2.1. Діаграма use-case

Незареєстрований клієнт може зареєструватися як клієнт або фермер.

Фермер на першому кроці додає склад та товар, котрий буде продавати. Після додавання товару та складу створюється сховище. Додавання товару автоматично створює заявку на продаж.

Клієнт може переглядати список товарів та додавати товари зі списку до кошика. Далі він може замовити товар з заявки на продаж від фермера чи купути наявний товар з залишків на складі фермера. Після додавання товару до кошика клієнту потрібно сплатити своє замовлення та обрати тип доставлення.

2.2. Опис застосованих математичних методів

У даному програмному продукті нема потреби використовувати математичні методи, оскільки більша частина роботи є зв’язком бази даних, створення записів у ній та пошук необхідних записів.

2.3. Опис використаної архітектури та шаблонів проєктування

Розроблений продукт є серверною частиною веб-додатка, що побудований з використанням клієнт-серверної архітектури.

Класична трирівнева клієнт серверна архітектура заснована на принципах розподілення структури додатків між трьома ярусами логічних обчислень:

- постачальниками ресурсу (серверами);
- середовищем зберігання даних (базами даних);
- споживачами ресурсу (клієнтами).

MVC

Model-View-Controller – схема розподілення даних програмного продукту, інтерфейсу користувача та керуючої логіки на три окремих компонента:

- Модель (Model) надає дані й реагує на команди контролера, змінюючи свій стан;

- Уявлення (View) відповідає за представлення даних моделі користувачеві, реагуючи на зміни моделі;
- Контролер (Controller) інтерпретує дії користувача, сповіщаючи модель про необхідність змін.

2.4 Опис моделей

Для функціонування розробленого продукту потрібна база даних, котра буде зберігати дані.

Дані у базі даних зберігаються у вигляді сутностей, котрі у розробляємому програмному продукті створюються автоматично при міграції моделей з файлу. Формат даних може бути строкою, числом, форматом дати з часом та у вигляді логічної змінної, котра зберігає у собі знчення True або False.

Список моделей програмного продукту:

- Користувач;
- Клієнт;
- Фермер;
- Персональні дані;
- Покупка;
- Продаж;
- Оплата;
- Тип оплати;
- Компанії доставлення;
- Тип доставлення;
- Продукт;
- Склад;
- Сховище;
- Угода;

Модель користувача

Модель користувача є стандартною моделлю фреймворку Django (див. Табл. 2.1).

Таблиця 2.1

Модель користувача

Назва колонки	Опис
id	Унікальний ідентифікатор кожного користувача у системі
password	Пароль користувача
last_login	Час останнього входу користувача у систему
is_superuser	Змінна, котра виділяє користувача до групи суперкористувачів, в котрої є більше прав
last_name	Призвіще користувача
email	Електронна пошта користувача
is_staff	Змінна, котра виділяє користувача до групи персоналу, в котрої є більше прав
is_active	Змінна, котра показує, чи активний обліковий запис користувача, чи ні
date_joined	Дата реєстрації користувача
first_name	Ім’я користувача

Модель клієнта

У цій моделі зберігаються ті користувачі, котрі зареєструвалися у додатку як клієнт. Модель створюється автоматично при реєстрації користувача у системі (див. Табл. 2.2).

Таблиця 2.2

Модель клієнта

Назва колонки	Опис
idClient	Унікальний ідентифікатор кожного клієнта
personalDataId	Ключ до моделі персональних даних
addres	Адреса клієнта, котра використовується для доставлення замовлення

Модель фермера

У цій моделі зберігаються ті користувачі, котрі зареєструвалися у додатку як фермер. Модель створюється автоматично при реєстрації користувача у системі (див. Табл. 2.3).

Таблиця 2.3

Модель фермера

Назва колонки	Опис
idFarmer	Унікальний ідентифікатор кожного фермера
personalDataId	Ключ до моделі персональних даних

Модель персональних даних

У цій моделі зберігаються особисті дані усіх користувачів. Модель створюється автоматично при реєстрації користувача у системі (див. Табл. 2.4).

Таблиця 2.4

Модель персональних даних

Назва колонки	Опис
idPersonalData	Унікальний ідентифікатор кожного запису особистих даних користувачів
isFarmer	Змінна, котра вказує, чи є користувач фермером
phoneNumber	Номер телефону користувача
money	Баланс користувача, котрий він може поповнювати/знімати кошти

Модель покупки

У цій моделі зберігаються заявки на покупку від клієнта. Модель створюється автоматично коли клієнт замовляє чи купує товар (див. Табл. 2.5).

Таблиця 2.5

Модель покупки

Назва колонки	Опис
idPurchase	Унікальний ідентифікатор кожної заявки на покупку
dateOfRequest	Дата створення заявки
dateOfPurchase	Дата запланованої покупки товару
amount	Кількість товару
clientId	Ключ до моделі клієнта, котрий оформив заявку на покупку
saleId	Ключ до моделі продажу, з котрої клієнт замовив товар

Модель продажу

У цій моделі зберігаються заявки на продаж від фермера. Модель створюється автоматично коли фермер додає товар до системи (див. Табл. 2.6).

Таблиця 2.6

Модель продажу

Назва колонки	Опис
idSale	Унікальний ідентифікатор кожної заявки на продаж
dateOfRequest	Дата створення заявки
dateOfHarvest	Дата збору врожаю
amount	Кількість товару
priceofGrowingHarvest	Вартість товару при замовленні на майбутнє
farmerId	Ключ до фермера, котрий додав товар
productId	Ключ до продукту, котрий додав фермер

Модель оплати

У цій моделі зберігаються дані оплати клієнтом свого замовлення. Створюється автоматично при оплаті товару клієнтом (див. Табл. 2.7).

Таблиця 2.7

Модель оплати

Назва колонки	Опис
idPayment	Унікальний ідентифікатор кожної оплати
dateOfPayment	Дата оплати
sum	Сума коштів
clientId	Ключ до клієнта, котрий оплатив товар
farmerId	Ключ до фермера, котрий продає товар

Продовж. табл. 2.7

typeOfPaymentId	Ключ до моделі типу оплати

Модель типу оплати

У цій моделі зберігаються дані по типу оплати. Оплата може буду повною чи частковою передоплатою. Ця модель створюється власноруч адміністратором системи (див. Табл. 2.8).

Таблиця 2.8

Модель типу оплати

Назва колонки	Опис
idTypeOfPayment	Унікальний ідентифікатор кожного типу оплати
name	Назва типу («Повна оплата» чи «Часткова передоплата»)

Модель компаній доставлення

У цій моделі зберігаються дані компаній з доставлення. Ця модель створюється та доповнюється власноруч адміністратором системи (див. Табл. 2.9).

Таблиця 2.9

Модель компаній доставлення

Назва колонки	Опис
idDeliveryCompanies	Унікальний ідентифікатор кожної компанії з доставлення
name	Назва компанії
priceForKm	Ціна доставлення за кожен кілометр дистанції

Модель типу доставлення

У цій моделі зберігаються дані по типу оплати. Доставлення може бути самовивозом чи доставлення однією з компаній. Ця модель створюється власноруч адміністратором системи (див. Табл. 2.10).

Таблиця 2.10

Модель типу доставлення

Назва колонки	Опис
idTypeOfDelivery	Унікальний ідентифікатор кожного типу доставлення
name	Назва типу («Самовивіз» чи «Доставка доставлення»)

Модель продукту

У цій моделі зберігаються дані товару, котрий додав фермер (див. Табл. 2.11).

Таблиця 2.11

Модель продукту

Назва колонки	Опис
idProduct	Унікальний ідентифікатор кожного товару
name	Назва продукту
unitOfMeasure	Одиниця виміру товару

Модель складу

У цій моделі зберігаються дані по складу фермера (див. Табл. 2.12).

Таблиця 2.12

Модель складу

Назва колонки	Опис
idStorage	Унікальний ідентифікатор кожного складу
addres	Адреса складу
farmerId	Ключ до фермера, котрому належить цей склад

Модель сховища

У цій моделі зберігаються дані по товару та складу, у котрому він зберігається. Модель створюється автоматично при додаванні фермером товару до системи (див. Табл. 2.13).

Таблиця 2.13

Модель сховища

Назва колонки	Опис
idStock	Унікальний ідентифікатор кожного сховища
productId	Ключ до товару, котрий зберігається у сховищі
amountOfProduct	Залишок товару у сховищі
dateOfHarvest	Дата збору врожаю
shelfLife	Термін придатності товару
price	Ціна зібраного товару
storageId	Ключ до складу, до якого прикріплено сховище

Модель угоди

У цій моделі зберігаються дані по угоді між клієнтом та фермером. Модель створюється автоматично при купівлі товару клієнтом у фермера.

Додаткові ключі до фермера, кілєнта та товару та строки price, amount потрібні в разі купівлі товару на поточний час (див. Табл. 2.14).

Таблиця 2.14

Модель угоди

Назва колонки	Опис
idTrade	Унікальний ідентифікатор кожної угоди
clientId	Ключ до клієнта, котрий купує товар
farmerId	Ключ до фермера, котрий продає товар
productId	Ключ до товару, котрий продається
purchaseId	Ключ до заявки на покупку
price	Ціна товару
amount	Кількість товару
typeOfPaymentId	Ключ до типу оплати
paymentId	Ключ до оплати
typeOfDeliveryId	Ключ до типу доставлення
deliveryCompaniesId	Ключ до компаній з доставлення
distance	Відстань від складу фермера до адре си клієнта

2.5. О пис структури програми та алгоритмів її функціонування

Розроблений програмний продукт розподілений по різнім файлам, структура котрих зображена нижче.

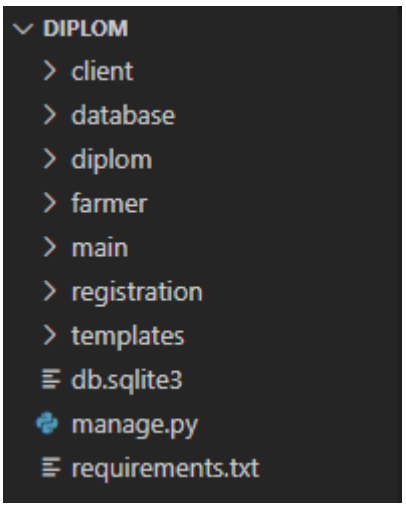

Рис. 2.2 Структура проекту

Проект складається з додатків, котрих у проєкті 6:

client;

database;

diplom;

farmer;

main;

registration.

А також каталога шаблонів HTML, бази даних, виконуючого файлу проекту та текстового документу з вимогами для роботи проетку.

Додаток складається з:

каталогу кешу байт-коду (__pycache__);

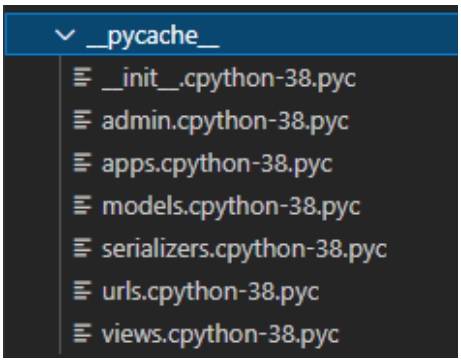

Рис. 2.3 Каталог міграцій

каталогу з міграціями до бази даних (migrations);

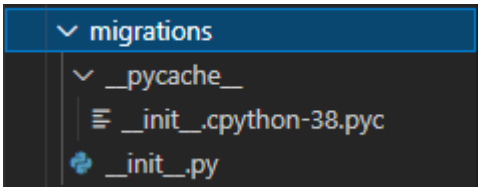

Рис. 2.4 Каталог кешу

файл позначення пакетів Python (__init__.py);

файл моделей адміністратора (admin.py);

файл конфігураціх додатку (apps.py);

файл з моделями бази даних (models.py);

файл з серіалайзерами моделей (serializers.py);

файл з тестами коду (tests.py);

файл маршрутизації додатку (urls.py);

файл з представленнями моделей (views.py)

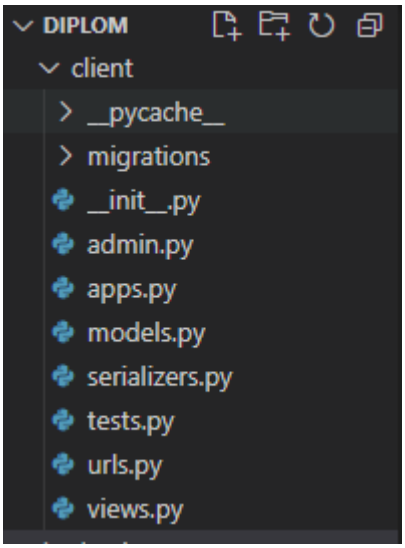

Рис. 2.5 Структура додатка

Інші додатки мають ту саму структуру, різниця лише в значеннях у файлах.

Проект має чотири основні додатки:

додаток бази даних (database);

додаток реєстрації (main);

додаток клієнта (client);

додаток фермера (farmer).

Знизу позначено основні класи та методи відповідно кожному файлу додатка.

База даних

models.py

class User(AbstractUser) – стандартна модель Django користувача .

class PersonalData(models.Model) – модель особистих даних.

class Client(models.Model) – модель клієнта.

class Farmer(models.Model) – модель фермера.

class TypeOfPayment(models.Model) – модель типів оплати.

class Payment(models.Model) – модель оплати.

class DeliveryCompanies(models.Model) – модель компаній з доставлення.

class TypeOfDelivery(models.Model) – модель типів доставлення.

class Product(models.Model) – модель товару.

class Storage(models.Model) – модель складу.

class Stock(models.Model) – модель сховища.

class Sale(models.Model) – модель заявки на продаж.

class Purchase(models.Model) – модель заявки на покупку.

class Trade(models.Model) – модель угоди.

Реєстрація

views.py

class UserAPI(generics.GenericAPIView) – API, що по GET-запиту надсилає список усіх клієнтів та по POST-запиту зберігає нового користувача та його особисті дані.


```
GET /
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "password": "pbkdf2_sha256$260000$irq5YnROu3I0Mj9yHUnepx$dHkleq0AQzWZoqRmmi4JsZ9Xt8ufT0veXQxwFMLpehQ=",
        "last_login": null,
        "is_superuser": false,
        "username": "Client",
        "first_name": "",
        "last_name": "",
        "email": "123",
        "is_staff": false,
        "is_active": true,
        "date_joined": "2021-06-08T09:14:48.440904Z",
```

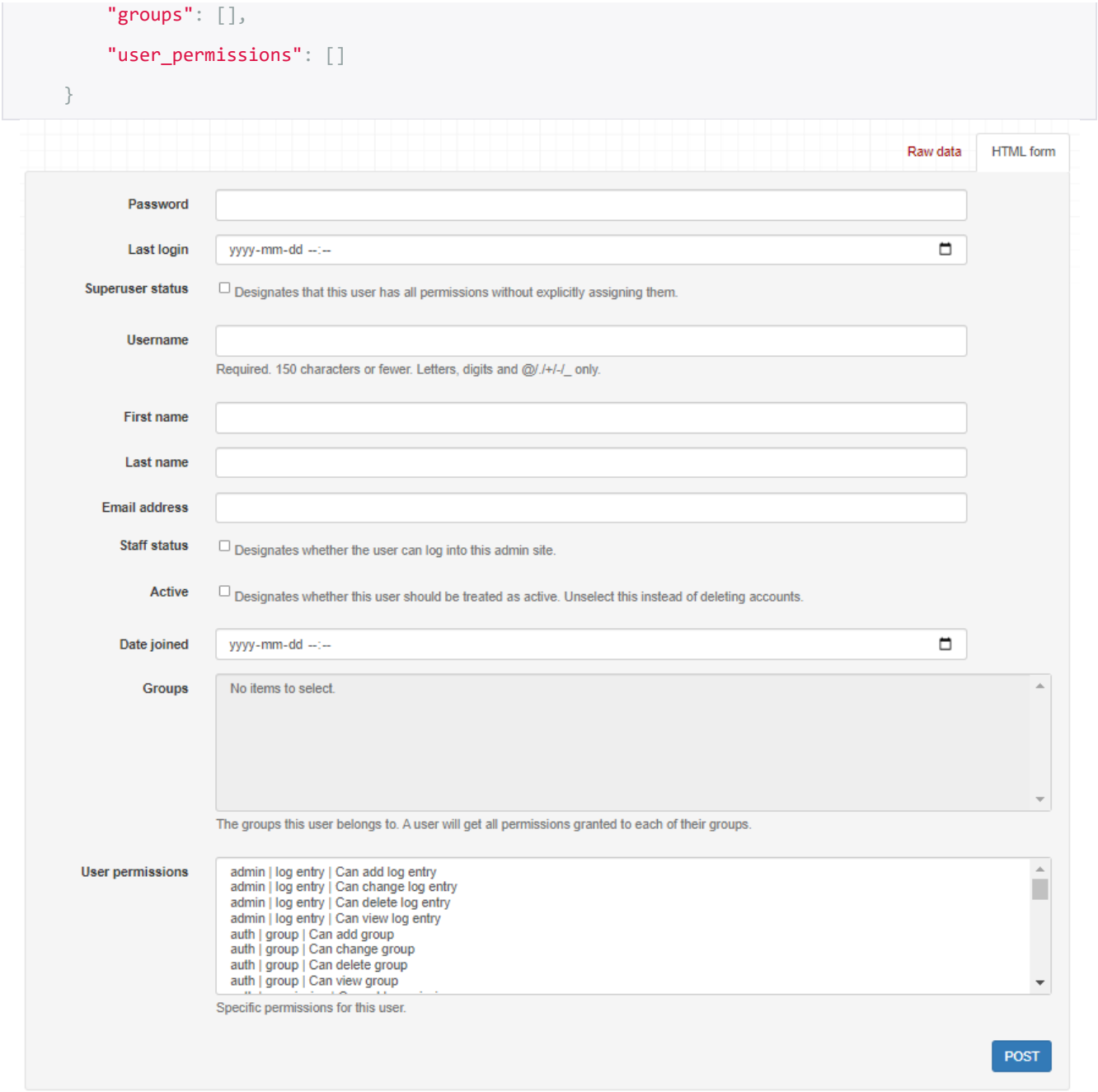

Рис. 2.6 Представлення UserAPI, POST запит

def main() – функція, котра виводить HTML сторінку.

Рис. 2.7 Представлення функції main при реєстрації клієнта

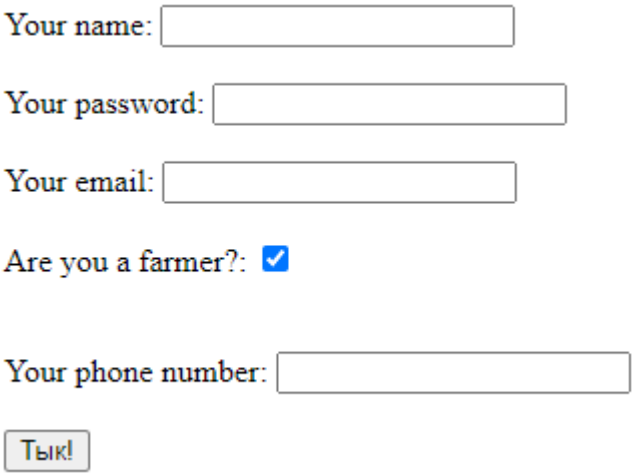

Рис. 2.8 Представлення функції main при реєстрації фермера

Клієнт

serializers.py

class FarmerPersonalDataSerializer(serializers.ModelSerializer) – окремий серіалайзер особистих даних для фермера.

class ProductSerializer(serializers.ModelSerializer) – серіалайзер моделі товара.

class FarmerSerializer(serializers.ModelSerializer) – серіалайзер моделі фермера.

class ClientSerializer(serializers.ModelSerializer) – серіалайзер моделі клієнта.

class PurchaseSerializer(serializers.ModelSerializer) – серіалайзер моделі заявок на покупку.

class TypeOfPaymentSerializer(serializers.ModelSerializer) – серіалайзер моделі типу оплати.

class PaymentSerializer(serializers.ModelSerializer) – серіалайзер моделі оплати.

class TypeOfDeliverySerializer(serializers.ModelSerializer) – серіалайзер моделі типу доставлення.

class DeliveryCompanySerializer(serializers.ModelSerializer) – серіалайзер моделі компаній з доставлення.

class PersonalDataSerializer(serializers.ModelSerializer) – серіалайзер моделі особистих даних з об’єктом серіалайзера ClientSerializer.

class ClientProfileSerializer(serializers.ModelSerializer) – серіалайзер моделі коритувача з об’єктом PersonalDataSerializer.

class SaleSerializer(serializers.ModelSerializer) – серіалайзер моделі заявки на продаж с об’єктами серіалайзерів FarmerSerializer та ProductSerializer.

class ClientBuySerializer(serializers.ModelSerializer) – серіалайзер моделі покупки з об’єктом серіалайзера SaleSerializer.

class ClientTradeSerializer(serializers.ModelSerializer) – серіалайзер моделі угоди з об’єктами серіалайзерів ProductSerializer, ClientSerializer, FarmerSerializer, PurchaseSerializer, TypeOfPaymentSerializer, PaymentSerializer, TypeOfDeliverySerializer та DeliveryCompanySerializer.

class ClientDeliverySerializer(serializers.ModelSerializer) – серіалайзер моделі типів доставленя з об’єктом серіалайзера ClientTradeSerializer.

class ClientBinSerializer(serializers.ModelSerializer) – серіалайзер моделі покупки з об’єктами серіалайзерів ClientSerializer та SaleSerializer.

class ClientPaySerializer(serializers.ModelSerializer) – окремий серіалайзер моделі покупки з об’єктами серіалайзерів ClientSerializer та SaleSerializer.

views.py

class ProductListAPI(generics.GenericAPIView) – це API для переглядання списку товарів.

```
GET /client/menu/
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "farmer": {
            "id": 1,
```

```
            "personalData": {
                "id": 2,
                "user": {
                    "id": 2,
                    "password": "pbkdf2_sha256$260000$GIqIsc7aAnEIUWvW4CNPQz$CpHqQumL4QggOq6KZoNU
LCjXoyupshm7ybygIwzP4nw=",
                    "last_login": null,
                    "is_superuser": false,
                    "username": "Farmer",
                    "first_name": "",
                    "last_name": "",
                    "email": "farmer@gmail.com",
                    "is_staff": false,
                    "is_active": true,
                    "date_joined": "2021-06-09T17:09:13.419091Z",
                    "groups": [],
                    "user_permissions": []
                },
                "isFarmer": true,
                "phoneNumber": "123123",
                "money": "0.00"
            }
        },
        "product": {
            "id": 1,
            "name": "Томат",
            "unitOfMeasure": "кг."
        },
        "dateOfRequest": null,
        "dateOfHarvest": null,
        "amount": "20.000",
        "price": "50.000"
    }
]
```


class ClientProfileAPI(generics.GenericAPIView) – це API для перегляданння та редагування профілю користувача.

```
GET /client/profile/
```

```
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "personaldata": {
            "id": 1,
            "client": {
                "id": 1,
                "addres": "123"
            },
            "isFarmer": false,
            "phoneNumber": "123",
            "money": "0.00"
        },
        "password": "pbkdf2_sha256$260000$irq5YnROu3I0Mj9yHUnepx$dHkleq0AQzWZoqRmmi4JsZ9Xt8ufT0veXQxwFMLpehQ=",
        "last_login": null,
        "is_superuser": false,
        "username": "Client",
        "first_name": "",
        "last_name": "",
        "email": "123",
        "is_staff": false,
        "is_active": true,
        "date_joined": "2021-06-08T09:14:48.440904Z",
        "groups": [],
        "user_permissions": []
    }
```

class ClientPayAPI(generics.GenericAPIView) – це API для поповнення балансу користувача.

```
GET /client/profile/pay/
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
```

```
Vary: Accept

[
    {
        "id": 1,
        "client": {
            "id": 1,
            "addres": "123"
        },
        "sale": {
            "id": 1,
            "farmer": {
                "id": 1,
                "personalData": {
                    "id": 2,
                    "user": {
                        "id": 2,
                        "password": "pbkdf2_sha256$260000$GIqIsc7aAnEIUWvW4CNPQz$CpHqQumL4QggOq6KZoNULCjXoyupshm7ybygIwzP4nw=",
                        "last_login": null,
                        "is_superuser": false,
                        "username": "Farmer",
                        "first_name": "",
                        "last_name": "",
                        "email": "farmer@gmail.com",
                        "is_staff": false,
                        "is_active": true,
                        "date_joined": "2021-06-09T17:09:13.419091Z",
                        "groups": [],
                        "user_permissions": []
                    },
                    "isFarmer": true,
                    "phoneNumber": "123123",
                    "money": "0.00"
                }
            },
            "product": {
                "id": 1,
                "name": "Томат",
                "unitOfMeasure": "кг."
            },
```

```
            "dateOfRequest": null,
            "dateOfHarvest": null,
            "amount": "20.000",
            "price": "50.000"
        },
        "dateOfRequest": null,
        "dateOfPurchase": null,
        "amount": "10.000"
    }
]
```


class ClientHistoryAPI(generics.GenericAPIView) – це API для перегляданння історії покупок клієнта.


```
GET /client/history/
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "product": {
            "id": 1,
            "name": "Томат",
            "unitOfMeasure": "кг."
        },
        "client": {
            "id": 1,
            "addres": "123"
        },
        "farmer": {
            "id": 1,
            "personalData": {
                "id": 2,
                "user": {
                    "id": 2,
```

```
                    "password": "pbkdf2_sha256$260000$GIqIsc7aAnEIUWvW4CNPQz$CpHqQumL4QggOq6KZoNU
LCjXoyupshm7ybygIwzP4nw=",
                    "last_login": null,
                    "is_superuser": false,
                    "username": "Farmer",
                    "first_name": "",
                    "last_name": "",
                    "email": "farmer@gmail.com",
                    "is_staff": false,
                    "is_active": true,
                    "date_joined": "2021-06-09T17:09:13.419091Z",
                    "groups": [],
                    "user_permissions": []
                },
                "isFarmer": true,
                "phoneNumber": "123123",
                "money": "0.00"
            }
        },
        "purchase": {
            "id": 1,
            "dateOfRequest": null,
            "dateOfPurchase": null,
            "amount": "10.000",
            "client": 1,
            "sale": 1
        },
        "typeOfPayment": {
            "id": 1,
            "name": "Полная оплата"
        },
        "payment": {
            "id": 1,
            "dateOfPayment": null,
            "sum": "500.000",
            "client": 1,
            "farmer": 1,
            "typeOfPayment": 1
        },
        "typeOfDelivery": {
            "id": 1,
```

```
            "name": "Самовывоз"
        },
        "deliveryCompany": {
            "id": 1,
            "name": "Glovo",
            "price": "70.000"
        },
        "price": "500.000",
        "amount": "10.000",
        "distance": "23.000"
    }
]
```


class ClientBinAPI(generics.GenericAPIView) – це API для переглядання та редагування кошика клієнта.


```
GET /client/bin/
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "client": {
            "id": 1,
            "addres": "123"
        },
        "sale": {
            "id": 1,
            "farmer": {
                "id": 1,
                "personalData": {
                    "id": 2,
                    "user": {
                        "id": 2,
                        "password": "pbkdf2_sha256$260000$GIqIsc7aAnEIUWvW4CNPQz$CpHqQumL4QggOq6K
ZoNULCjXoyupshm7ybygIwzP4nw=",
```

```
                        "last_login": null,
                        "is_superuser": false,
                        "username": "Farmer",
                        "first_name": "",
                        "last_name": "",
                        "email": "farmer@gmail.com",
                        "is_staff": false,
                        "is_active": true,
                        "date_joined": "2021-06-09T17:09:13.419091Z",
                        "groups": [],
                        "user_permissions": []
                    },
                    "isFarmer": true,
                    "phoneNumber": "123123",
                    "money": "0.00"
                }
            },
            "product": {
                "id": 1,
                "name": "Томат",
                "unitOfMeasure": "кг."
            },
            "dateOfRequest": null,
            "dateOfHarvest": null,
            "amount": "20.000",
            "price": "50.000"
        },
        "dateOfRequest": null,
        "dateOfPurchase": null,
        "amount": "10.000"
    }
]
```


class ClientBuyAPI(generics.GenericAPIView) – це API для покупки товарів.

```
GET /client/buy/
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
```

```
Vary: Accept

[
    {
        "id": 1,
        "sale": {
            "id": 1,
            "farmer": {
                "id": 1,
                "personalData": {
                    "id": 2,
                    "user": {
                        "id": 2,
                        "password": "pbkdf2_sha256$260000$GIqIsc7aAnEIUWvW4CNPQz$CpHqQumL4QggOq6KZoNULCjXoyupshm7ybygIwzP4nw=",
                        "last_login": null,
                        "is_superuser": false,
                        "username": "Farmer",
                        "first_name": "",
                        "last_name": "",
                        "email": "farmer@gmail.com",
                        "is_staff": false,
                        "is_active": true,
                        "date_joined": "2021-06-09T17:09:13.419091Z",
                        "groups": [],
                        "user_permissions": []
                    },
                    "isFarmer": true,
                    "phoneNumber": "123123",
                    "money": "0.00"
                }
            },
            "product": {
                "id": 1,
                "name": "Томат",
                "unitOfMeasure": "кг."
            },
            "dateOfRequest": null,
            "dateOfHarvest": null,
            "amount": "20.000",
            "price": "50.000"
```

```
        },
        "dateOfRequest": null,
        "dateOfPurchase": null,
        "amount": "10.000",
        "client": 1
    }
]
```


class TypeOfDeliveryAPI(generics.GenericAPIView) – це API для вибору типу доставлення товару.


```
GET /client/buy/delivery
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "trade": {
            "id": 1,
            "product": {
                "id": 1,
                "name": "Томат",
                "unitOfMeasure": "кг."
            },
            "client": {
                "id": 1,
                "addres": "123"
            },
            "farmer": {
                "id": 1,
                "personalData": {
                    "id": 2,
                    "user": {
                        "id": 2,
                        "password": "pbkdf2_sha256$260000$GIqIsc7aAnEIUWvW4CNPQz$CpHqQumL4QggOq6K
ZoNULCjXoyupshm7ybygIwzP4nw=",
```

```
                "last_login": null,
                "is_superuser": false,
                "username": "Farmer",
                "first_name": "",
                "last_name": "",
                "email": "farmer@gmail.com",
                "is_staff": false,
                "is_active": true,
                "date_joined": "2021-06-09T17:09:13.419091Z",
                "groups": [],
                "user_permissions": []
            },
            "isFarmer": true,
            "phoneNumber": "123123",
            "money": "0.00"
        }
    },
    "purchase": {
        "id": 1,
        "dateOfRequest": null,
        "dateOfPurchase": null,
        "amount": "10.000",
        "client": 1,
        "sale": 1
    },
    "typeOfPayment": {
        "id": 1,
        "name": "Полная оплата"
    },
    "payment": {
        "id": 1,
        "dateOfPayment": null,
        "sum": "500.000",
        "client": 1,
        "farmer": 1,
        "typeOfPayment": 1
    },
    "typeOfDelivery": {
        "id": 1,
        "name": "Самовывоз"
```

```
            },
            "deliveryCompany": {
                "id": 1,
                "name": "Glovo",
                "price": "70.000"
            },
            "price": "500.000",
            "amount": "10.000",
            "distance": "23.000"
        },
        "name": "Самовывоз"
    },
    {
        "id": 2,
        "trade": null,
        "name": "Доставка агригатором"
    }
]
```


Фермер

serializers.py

class FarmerSerializer(seriaizers.ModelSerializer) – серіалайзер моделі фермера.

class FarmerPersonalDataSerializer(seriaizers.ModelSerializer) – окремий серіалайзер моделі особистих даних для фермера.

class FarmerStorageSerializer(seriaizers.ModelSerializer) – серіалайзер моделі фермера з об’єктом серіалайзера FarmerPersonalDataSerializer.

class StorageSerializer(seriaizers.ModelSerializer) – серіалайзер моделі складу з об’єктом серіалайзера FarmerStorageSerializer.

class StockSerializer(seriaizers.ModelSerializer) – серіалайзер моделі сховища з об’єктом серіалайзера StorageSerializer.

class ProductSerializer(seriaizers.ModelSerializer) – серіалайзер моделі товару з об’єктом серіалайзера StockSerializer.

class ClientSerializer(seriaizers.ModelSerializer) – серіалайзер моделі клієнта.

class PurchaseSerializer(seriaizers.ModelSerializer) – серіалайзер моделі покупки.

class TypeOfPaymentSerializer(seriaizers.ModelSerializer) – серіалайзер моделі типу оплати.

class PaymentSerializer(seriaizers.ModelSerializer) – серіалайзер моделі оплати.

class TypeOfDeliverySerializer(seriaizers.ModelSerializer) – серіалайзер моделі типу доставлення.

class DeliveryCompanySerializer(seriaizers.ModelSerializer) – серіалайзер моделі компаній з доставлення.

class PersonalDataSerializer(seriaizers.ModelSerializer) – серіалайзер моделі особистих даних з об’єктом серіалайзера FarmerSerializer.

class FarmerProfileSerializer(seriaizers.ModelSerializer) – серіалайзер моделі користувача з об’єктом серіалайзера PersonalDataSerializer.

class FarmerPaySerializer(seriaizers.ModelSerializer) – серіалайзер моделі особистих даних.

class FarmerHistorySerializer(seriaizers.ModelSerializer) – серіалайзер моделі угоди з об’єктами серіалайзерів ProductSerializer, ClientSerializer, FarmerSerializer, PurchaseSerializer, TypeOfPaymentSerializer та DeliveryCompanySerializer.

class ProductAddSerializer(seriaizers.ModelSerializer) – серіалайзер моделі товару з об’єктом серіалайзеру StockSerializer.

class ProductEditSerializer(seriaizers.ModelSerializer) – серіалайзер моделі товару з об’єктом серіалайзеру StockSerializer.

views.py

class FarmerProfileAPI(generics.GenericAPIView) – це API для переглядання та редагування профілю фермера.

```
GET /farmer/profile/
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "personaldata": {
            "id": 1,
            "farmer": null,
            "isFarmer": false,
            "phoneNumber": "123",
```

```
            "money": "0.00"
        },
        "password": "pbkdf2_sha256$260000$irq5YnROu3I0Mj9yHUnepx$dHkleq0AQzWZoqRmmi4JsZ9Xt8ufT0veXQxwFMLpehQ=",
        "last_login": null,
        "is_superuser": false,
        "username": "Client",
        "first_name": "",
        "last_name": "",
        "email": "123",
        "is_staff": false,
        "is_active": true,
        "date_joined": "2021-06-08T09:14:48.440904Z",
        "groups": [],
        "user_permissions": []
    }
```


class FarmerPayAPI(generics.GenericAPIView) – це API для переглядання й виведення коштів з балансу.


```
GET /farmer/profile/pay/
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "isFarmer": false,
        "phoneNumber": "123",
        "money": "0.00",
        "user": 1
    }
```


class FarmerHistoryAPI(generics.GenericAPIView) – це API для переглядання історії продаж фермера.

```
GET /farmer/history/
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "product": {
            "id": 1,
            "stock": {
                "id": 1,
                "storage": {
                    "id": 1,
                    "farmer": {
                        "id": 1,
                        "personalData": {
                            "id": 2,
                            "isFarmer": true,
                            "phoneNumber": "123123",
                            "money": "0.00",
                            "user": 2
                        }
                    },
                    "addres": "Парк Шевченко"
                },
                "amountOfProduct": "213.000",
                "dateOfHarvest": null,
                "shelfLife": null,
                "price": "50.000",
                "product": 1
            },
            "name": "Томат",
            "unitOfMeasure": "кг."
        },
        "client": {
            "id": 1,
            "addres": "123",
            "personalData": 1
```

```
        },
        "farmer": {
            "id": 1
        },
        "purchase": {
            "id": 1,
            "dateOfRequest": null,
            "dateOfPurchase": null,
            "amount": "10.000",
            "client": 1,
            "sale": 1
        },
        "typeOfPayment": {
            "id": 1,
            "name": "Полная оплата"
        },
        "payment": {
            "id": 1,
            "dateOfPayment": null,
            "sum": "500.000",
            "client": 1,
            "farmer": 1,
            "typeOfPayment": 1
        },
        "typeOfDelivery": {
            "id": 1,
            "name": "Самовывоз"
        },
        "deliveryCompany": {
            "id": 1,
            "name": "Glovo",
            "price": "70.000"
        },
        "price": "500.000",
        "amount": "10.000",
        "distance": "23.000"
    }
]
```

class StorageAPI(generics.GenericAPIView) – це API для перегляданння списку складів фермера.

```
GET /farmer/storages/
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "farmer": {
            "id": 1,
            "personalData": {
                "id": 2,
                "isFarmer": true,
                "phoneNumber": "123123",
                "money": "0.00",
                "user": 2
            }
        },
        "addres": "Парк Шевченко"
    }
]
```

class StorageEditAPI(generics.GenericAPIView) – це API для додавання, редагування та видалення складів.

```
GET /farmer/storages/edit/
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {       "id": 1,
        "farmer": {
            "id": 1,
```

```
            "personalData": {
                "id": 2,
                "isFarmer": true,
                "phoneNumber": "123123",
                "money": "0.00",
                "user": 2
            }
        },
        "addres": "Парк Шевченко"
    }
]
```


class ProductListAPI(generics.GenericAPIView) – це API для перегляду списку товарів фермера.


```
GET /farmer/products/
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "stock": {
            "id": 1,
            "storage": {
                "id": 1,
                "farmer": {
                    "id": 1,
                    "personalData": {
                        "id": 2,
                        "isFarmer": true,
                        "phoneNumber": "123123",
                        "money": "0.00",
                        "user": 2
                    }
                },
                "addres": "Парк Шевченко"
```

```
            },
            "amountOfProduct": "213.000",
            "dateOfHarvest": null,
            "shelfLife": null,
            "price": "50.000",
            "product": 1
        },
        "name": "Томат",
        "unitOfMeasure": "кг."
    }
]
```


class ProductAddAPI(generics.GenericAPIView) – це API для додавання нових товарів фермером.


```
GET /farmer/products/add/
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "stock": {
            "id": 1,
            "storage": {
                "id": 1,
                "farmer": {
                    "id": 1,
                    "personalData": {
                        "id": 2,
                        "isFarmer": true,
                        "phoneNumber": "123123",
                        "money": "0.00"
                        "user": 2
                    }
                },
                "addres": "Парк Шевченко"
```

```
            },
            "amountOfProduct": "213.000",
            "dateOfHarvest": null,
            "shelfLife": null,
            "price": "50.000",
            "product": 1
        },
        "name": "Томат",
        "unitOfMeasure": "кг."
    }
]
```


class ProductEditAPI(generics.GenericAPIView) – це API для редагування наявних товарів фермером.


```
GET /farmer/products/edit/
HTTP 200 OK
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
    {
        "id": 1,
        "stock": {
            "id": 1,
            "storage": {
                "id": 1,
                "farmer": {
                    "id": 1,
                    "personalData": {
                        "id": 2,
                        "isFarmer": true,
                        "phoneNumber": "123123",
                        "money": "0.00",
                        "user": 2
                    }
                },
                "addres": "Парк Шевченко"
```

```
            },
            "amountOfProduct": "213.000",
            "dateOfHarvest": null,
            "shelfLife": null,
            "price": "50.000",
            "product": 1
        },
        "name": "Томат",
        "unitOfMeasure": "кг."
    }
]
```


2.6 Опис використаних технологій та мов програмування

Python

Це високорівнева мова програмування зі строгою типізацією, призначена для спрощення роботи розробника.

Перелік використаних бібліотек:

- Django 3.2;
- django-extensions 3.1.3;
- django-filter 2.4.0;
- djangorestframework 3.12.4;
- dgangorestframework-simplejwt 4.7.0;
- jwt 1.2.0;
- PyJWT 2.1.0;
- PyMySQL 1.0.2.

Django

Вільний фреймворк для вебдодатків на мові Python.

Цей фреймворк значно спрощує роботу розробника, оскільки він бере на себе більшу частину роботи, таку як:

Маршрутизація;

HTTP запити;

Міграція моделей до бази даних.

Rest framework

Це потужний та гнучкий інструментарій для побудови веб-API.

Цей фреймворк спрощує створення API для Django, оскільки більшість класів (API) наслідується від його класів, також цей фреймворк дозволяє створювати серіалізатори для зручного використання бази даних та має простий графічний інтерфейс для отримання та посилання запитів.

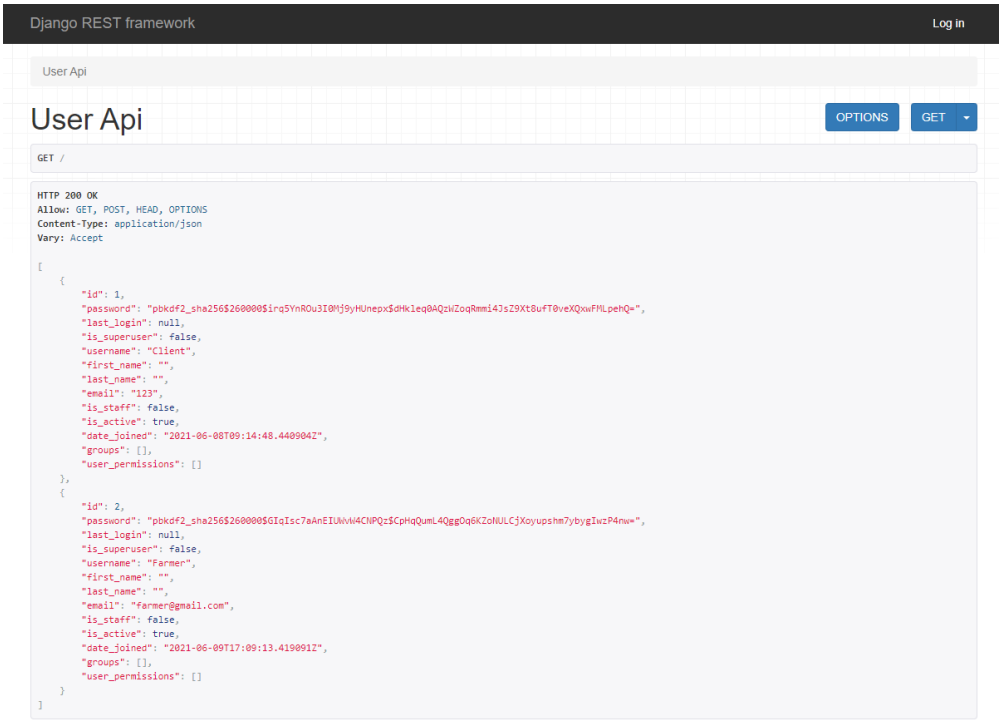

Рис. 2.9 Графічне уявлення GET-запиту

Обробивши цей запит, сервер надсилає адміністратору системи список усіх користувачів у форматі JSON. JSON (JavaScript Object Notation) – текстовий формат для обміну даними, схожий на стандартний словник python. Використовується через його зрозумілість та простоту.

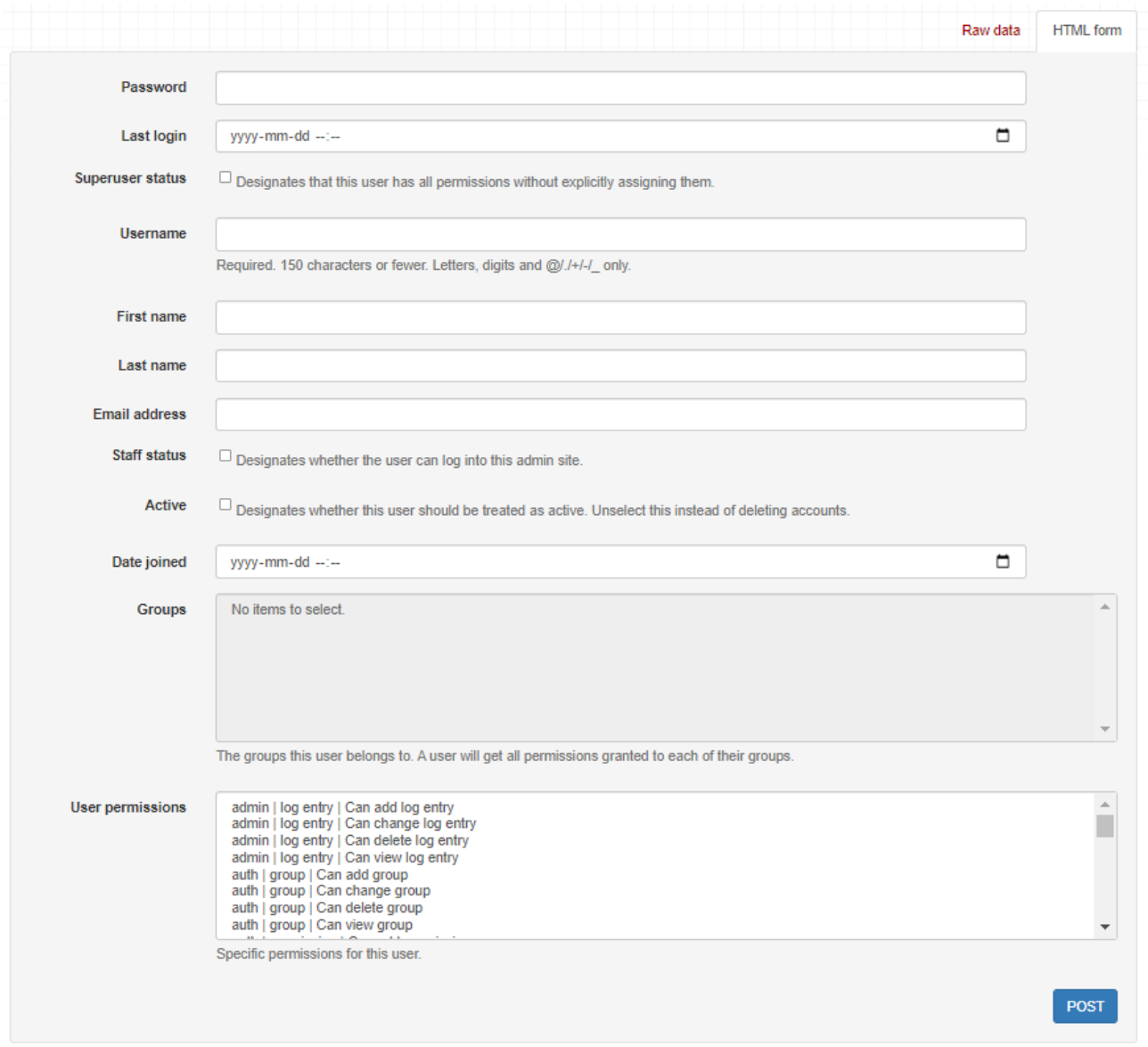

Рис. 2.10 Графічне уявлення POST-запиту

На рис. 2.10 зверху зображена HTML форма запиту.

HTML (HyperText Markup Language) – мова розмітки документів. Дозволяє створювати документи та розміщати на них певні блоки (теги) для зручного використання.

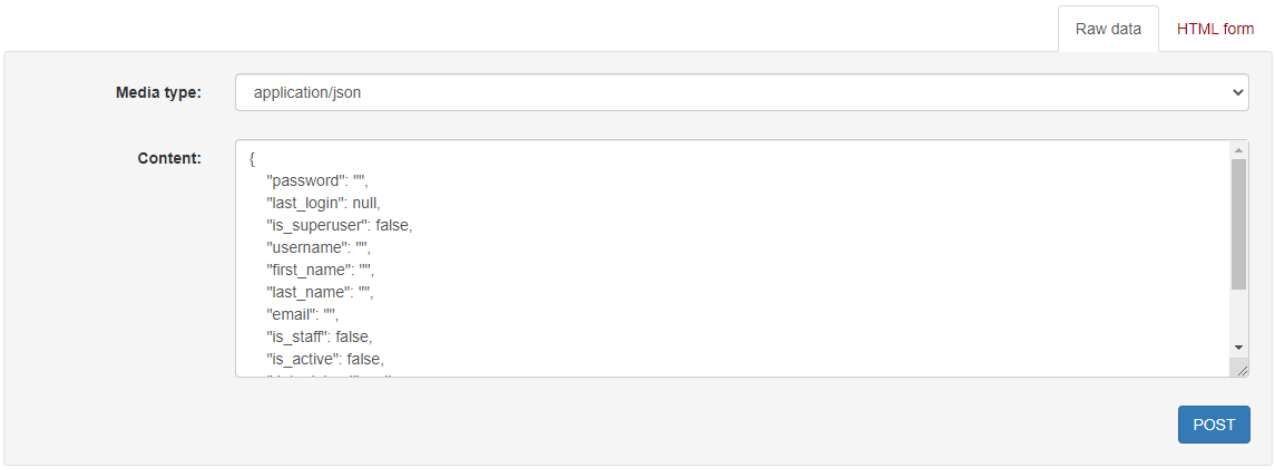

Рис. 2.11 Уявлення POST-запиту у форматі JSON

На рис. 2.11 зображено JSON об’єкт, котрий передається серверу.

Insomnia

Insomnia – клієнт з відкритим кодом для посилання запитів.

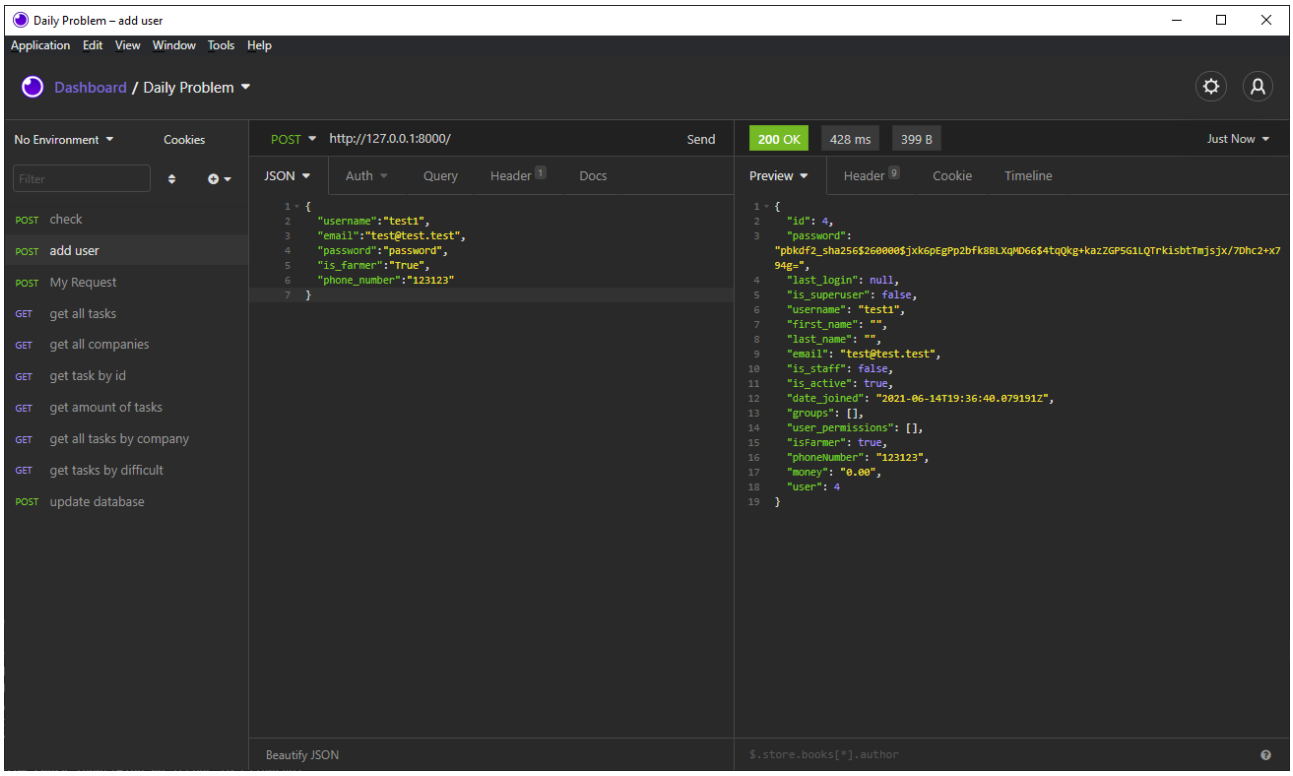

Рис. 2.12 Відправка POST-запиту (зліва) та отримання відповіді (справа)

Sqllite

Спрощений варіант мови SQL.

SQLiteStudio

Проста СУБД для роботи з базами даних.

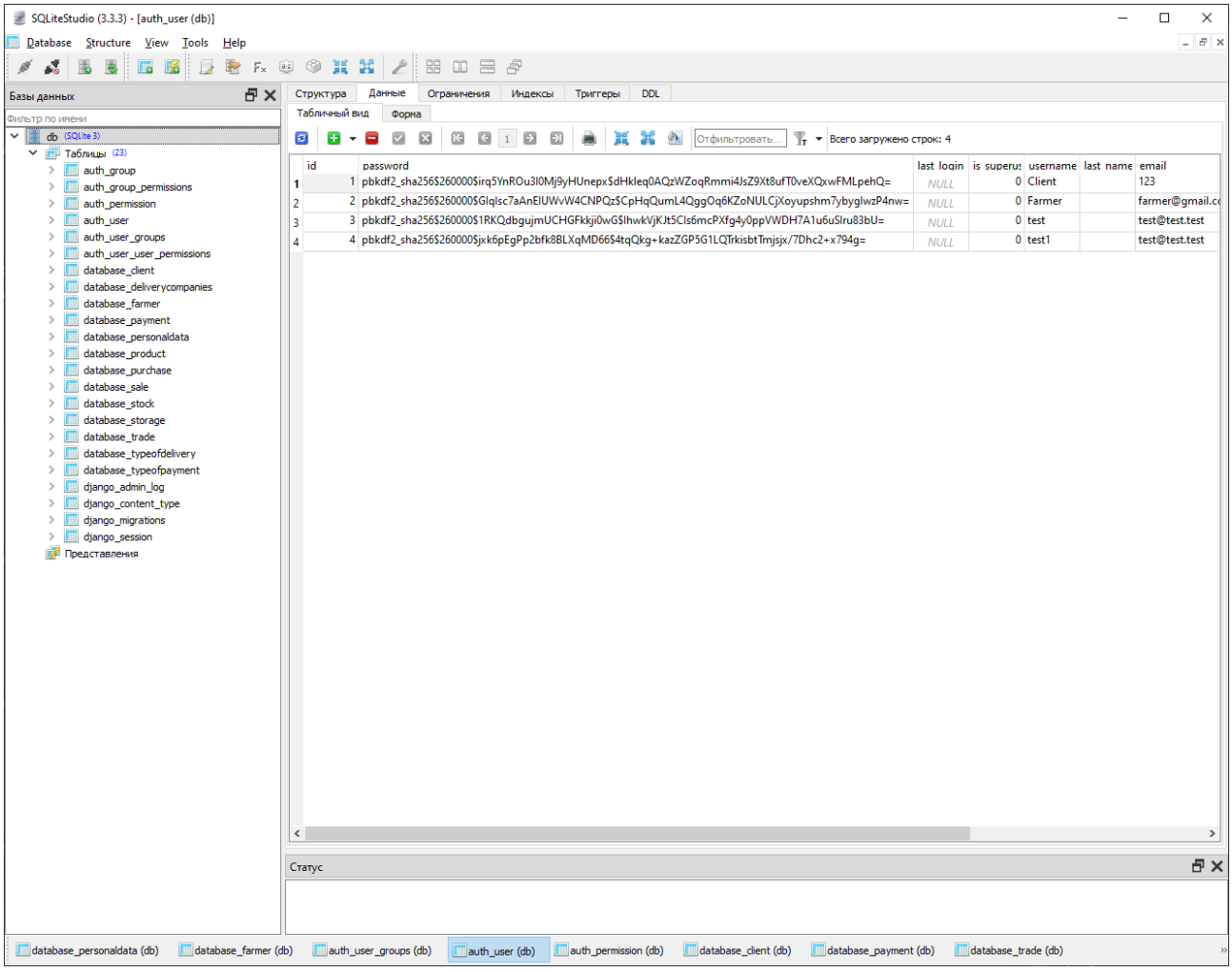

Рис. 2.13 Графічний інтерфейс SQLiteStudio

Simple JWT

Бібліотека формування токенів для захищеної авторизації

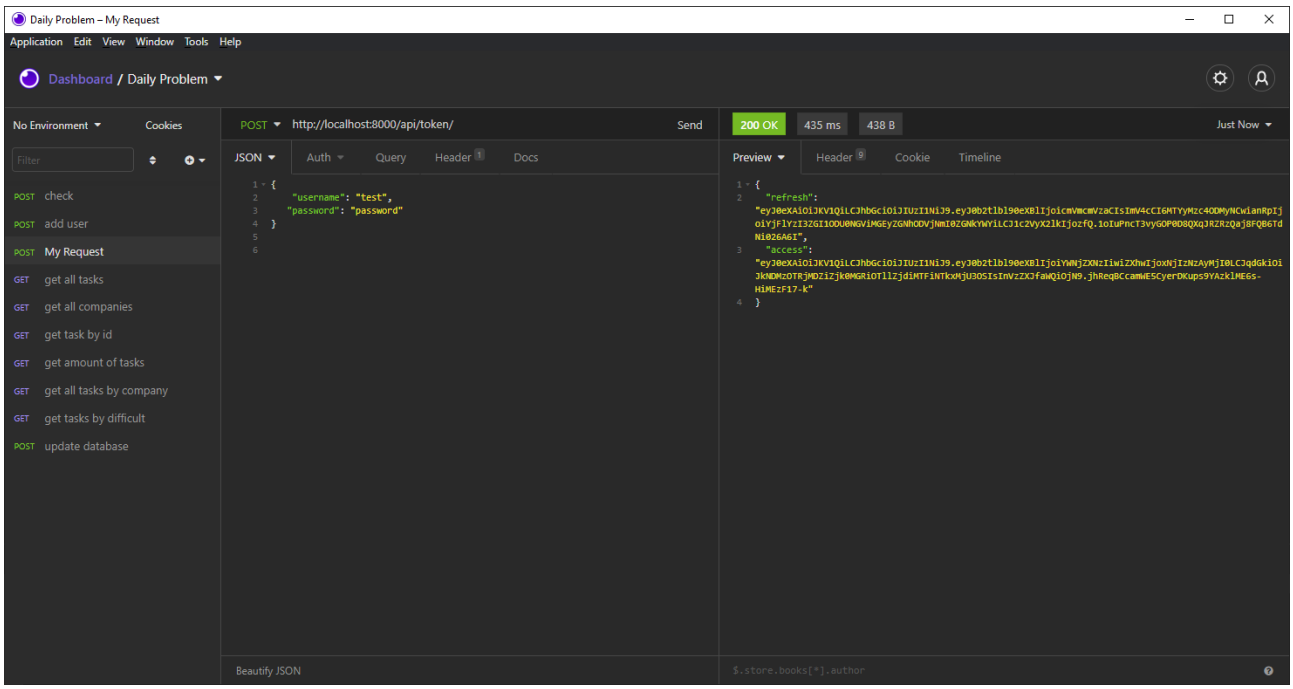

Рис. 2.14 Запит на отримання токену клієнта

2.7. Обґрунтування та організація вхідних та вихідних даних програми

Вхідними даними застосунку, що розробляється є дані, які вводяться користувачем з клавіатури у спеціальні екранні форми для вводу. На цей момент введення даних реалізовано через передавання JSON об’єкту POST запитом.

Все вхідні дані валідуються та обробляються стандартними методами Django.

Вхідними даними системи для клієнта є:

– дані форми реєстрації;
– дані форми входу;
– дані профілю;
– дані форми покупки;
– дані форми оплати;
– дані форми вибору методу доставлення.

Вхідніми даними системи для фермера є:

– дані форми реєстрації;
– дані форми входу;
– дані профілю;

– дані форми додавання товару;
– дані форми редагування товару;
– дані складів;
– дані форми додавання складу;
– дані форми редагування складу.

Вхідними даними адміністратора є усі вхідні дані клієнта та фермера.

Усі вищеперераховані дані являються текстовими окрім аватара користувача у профілі та зображень товарів, що додає до системи фермер, вони зберігаються у вигляді файлів.

Вихідними даними системи для клієнта є:

– дані профілю;
– дані товарів;
– дані історії покупки;
– дані фермера;
– дані кошика.

Вихідними даними системі для фермера є:

– дані профілю;
– дані товарів;
– дані історії продаж;
– дані клієнта;
– дані складів;
– дані угоди.

Вихідними даними адміністратора є усі вихідні дані клієнта та фермера.

2.8. Опис розробленого програмного продукту

2.8.1. Використані технічні засоби

Для запуску розробленої програми можна використовувати будь-який пристрій, на якому є доступ до мережі інтернет та на який можна встановити

браузер. Це може бути як мобільний телефон так і персональний комп’ютер.

Для запуску сервера потрібна система Windows чи Linux.

2.8.2. Використані програмні засоби

Для функціонування розробленого програмного продукту можна використовувати будь-яку операційну систему, яка має стабільне підключення до мережі Інтернет та яка підтримує мову програмування Python та на неї можна встановити Django.

2.8.3. Виклик та завантаження програми

Для запуску сервера потрібно:

1. Перейти до каталогу з проектом.
2. Запустити виконуючий файл за допомогою командного рядка та команди “python manage.py runserver” та натиснути Enter.

2.8.4 Опис інтерфейсу користувача

Розробляємий програмний продукт є серверною частиною веб-додатку, тому графічного інтерфейсу не має.

РОЗДІЛ 3
ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки програмного продукту

Початкові дані:

передбачуване число операторів програми – 1800;

коефіцієнт складності програми – 2,0;

коефіцієнт корекції програми в ході її розробки – 0,1;

годинна заробітна плата програміста – 89 грн/год;

Середньо українська заробітна плата junior-програміста на Python, складає близько 575 доларів у місяць. При курсі валют НБУ на початок червня 2021 року один американський долар дорівнює 27,1 грн, тому середня зарплата в гривнях дорівнює 15626 грн. При восьмигодинному робочому дні (176 годин в місяць в середньому) середня зарплата за годину буде становити 89 грн.

(https://ain.ua/2021/02/17/obzor-zarplat-ain-ua-2021-2/)

коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі – 1,2;

коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 0,8;

вартість машино-години ЕОМ – 15 грн/год.

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:

$$t = t_o + t_u + t_a + t_n + t_{отл} + t_д \text{, людино-годин,} \qquad (3.1)$$

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

$t_и$ – витрати праці на дослідження алгоритму рішення задачі;

$t_а$ – витрати праці на розробку блок-схеми алгоритму;

$t_п$ – 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):

$$Q = q \cdot C \cdot (1 + p), \text{ де} \qquad (3.2)$$

q – передбачуване число операторів;

С – коефіцієнт складності програми;

р – коефіцієнт кореляції програми в ході її розробки.

$$Q = 1800 \cdot 2{,}0 \cdot (1 + 0{,}1) = 3272{,}72;$$

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

$$t_u = \frac{Q \cdot B}{(75 \ldots 85) \cdot K}, \text{ людино-годин,} \qquad (3.3)$$

де В – коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі;

К – коефіцієнт кваліфікації програміста, обумовлений стажем роботи з даної спеціальності;

$t_u = \frac{3272,72 \cdot 1,2}{85 \cdot 0,8} = 57,75$, людино-годин.

Витрати праці на розробку алгоритму рішення задачі:

$$t_a = \frac{Q}{(20...25) \cdot K}; \qquad (3.4)$$

$t_a = \frac{3272,72}{25 \cdot 0,8} = 163,63$, людино-годин.

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25) \cdot K}; \qquad (3.5)$$

$t_n = \frac{3272,72}{25 \cdot 0,8} = 163,63$, людино-годин.

Витрати праці на налагодження програми на ЕОМ:
за умови автономного налагодження одного завдання:

$$t_{отл} = \frac{Q}{(4...5) \cdot K}; \qquad (3.6)$$

$t_n = \frac{3272,72}{5 \cdot 0,8} = 818,18$, людино-годин,

за умови комплексного налагодження завдання:

$$t_{отл}^{к} = 1,5 \cdot t_{отл}; \qquad (3.7)$$

$t_{отл}^{к} = 1,5 \cdot 818,18 = 1227,27$, людино-годин

Витрати праці на підготовку документації:

$$t_{д} = t_{др} + t_{до};$$ (3.8)

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

$$tдр = \frac{Q}{(15...20)\cdot К};$$ (3.9)

$$tдр = \frac{3272,72}{20 \cdot 0,8} = 204,54$$ людино-годин.

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

$$t_{до} = 0,75 \cdot t_{др};$$ (3.10)

$$t_{до} = 0,75 \cdot 204,54 = 153,41,$$ людино-годин.

$$t_{д} = 204,54 + 153,41 = 357,95$$, людино-годин.

Отримаємо трудомісткість розробки програмного забезпечення:

t = 50 + 57,75 + 163,63 + 163,63 + 818,18 + 357,95 = 1611,14 , людино-годин.

У результаті ми розрахували, що в загальній складності необхідно 1611,14 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 Зз/п і витрат машинного часу, необхідного на налагодження програми на ЕОМ.

$$K_{ПО} = З_{ЗП} + З_{МВ}, \text{ грн,} \qquad (3.11)$$

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

$$З_{ЗП} = t \cdot C_{ПР}, \text{ грн,} \qquad (3.12)$$

де t – загальна трудомісткість, людино-годин;

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

$З_{ЗП} = 1611{,}14 \cdot 89 = 143391{,}46$, грн.

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

$$З_{МВ} = t_{отл} \cdot C_{М}, \text{ грн,} \qquad (3.13)$$

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

$С_{МЧ}$ – вартість машино-години ЕОМ, грн/год.

$З_{МВ} = 818{,}18 \cdot 15 = 12272{,}7$ грн.

$К_{ПО} = 143391{,}46 + 12272{,}7 = 155664{,}16$ грн.

Очікуваний період створення ПЗ:

$$T = \frac{t}{B_k \cdot F_p}, \text{ мес.} \qquad (3.14)$$

де B_k- число виконавців;

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

$$T = \frac{1611,14}{1 \cdot 176} = 9,1 \text{ міс.}$$

Висновки: були визначені витрати на створення ПЗ, зазначеного у технічному завданні кваліфікаційної роботи – 155664,16 грн. і час створення програмного забезпечення – 9,1 місяців.

ВИСНОВКИ

В результаті виконання даної кваліфікаційної роботи була досягнена мета, поставлена на початку роботи – була створена серверна частина веб-додатку торговельно-інформаційної системи, що реалізує спроможність звичайного користувача купувати свіжі овочі чи вирощувати їх власноруч.

Актуальність розробленого продукту обумовлена відсутністю конкурентів на ринку та інноваційною ідеєю, котра може змінити ринок харчових продуктів.

Розроблений веб-додаток розроблявся для приватної системи.

Дане програмне забезпечення надає серверну частину торговельно-інформаційної системи для проєкту, котрий реалізує онлайн купівлю та продаж товарів.

Була розроблена архітектура проєкту, структура програми являє собою серверну частину до вебклієнта, написану на основі фреймворків Django та Django Rest Framework, що використовує мову програмування Python. Для реалізації було застосовано цілий ряд сторонніх бібліотек, що значно спрощують роботу.

У майбутньому планується покращення та збільшення функціональності програмного продукту, зокрема його зв’язок з клієнтською частиною.

Також у даній кваліфікаційній роботі в «Економічному розділі» було визначено трудомісткість розробки програмного забезпечення (1611 чол-год), були підраховані витрати на створення програмного забезпечення (155664 грн) і очікуваний період розробки (9.1 міс).

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Python's documentation. URL: https://www.python.org/doc/
2. Django documentation. URL: https://docs.djangoproject.com/en/3.2/
3. Library documentation. URL: https://stackoverflow.com/
4. Description of structures. URL: https://habr.com/
5. Rest Framework documentation. URL: https://www.django-rest-framework.org/tutorial/quickstart/
6. Simple JWT documentation. URL: https://django-rest-framework-simplejwt.readthedocs.io/en/latest/
7. Лутц М. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с.
8. Златопольский Д.М. Основы программирования на языке Python. – М.: ДМК Пресс, 2017. – 284 с.
9. Лутц М. Программирование на Python, том I, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.
10. Лутц М. Программирование на Python, том II, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 992 с.
11. Гэддис Т. Начинаем программировать на Python. – 4-е изд.: Пер. с англ. – СПб.: БХВ-Петербург, 2019. – 768 с.
12. Лучано Рамальо Python. К вершинам мастерства. – М.: ДМК Пресс, 2016. – 768 с.
13. Свейгарт, Эл. Автоматизация рутиных задач с помощью Python: практическое руководство для начинающих. Пер. с англ. — М.: Вильямс, 2016. – 592 с.
14. Рейтц К., Шлюссер Т. Автостопом по Python. – СПб.: Питер, 2017. – 336 с.
15. Любанович Билл Простой Python. Современный стиль программирования. – СПб.: Питер, 2016. – 480 с.: – (Серия «Бестсепперы O’Reilly»).

16. Федоров, Д. Ю. Программирование на языке высокого уровня Python : учебное пособие для прикладного бакалавриата / Д. Ю. Федоров. – 2-е изд., перераб. и доп. – Москва : Издательство Юрайт, 2019. – 161 с
17. Шелудько, В. М. Основы программирования на языке высокого уровня Python: учебное пособие / В. М. Шелудько. – Ростов-на-Дону, Таганрог: Издательство Южного федерального университета, 2017. – 146 с.
18. Шелудько, В. М. Язык программирования высокого уровня Python. Функции, структуры данных, дополнительные модули: учебное пособие / В. М. Шелудько. – Ростов-на-Дону, Таганрог: Издательство Южного федерального университета, 2017. – 107 с.
19. Доусон М. Программируем на Python. – СПб.: Питер, 2014. – 416 с.
20. Прохоренок Н.А. Python 3 и PyQt. Разработка приложений. – СПб.: БХВ-Петербург, 2012. – 704 с.
21. Пилгрим Марк. Погружение в Python 3 (Dive into Python 3 на русском)
22. Прохоренок Н.А. Самое необходимое. — СПб.: БХВ-Петербург, 2011. — 416 с.

ДОДАТОК А

КОД ПРОГРАМИ

diplom

settings.py

```
"""
Django settings for diplom project.

Generated by 'django-admin startproject' using Django 3.2.

For more information on this file, see
https://docs.djangoproject.com/en/3.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/3.2/ref/settings/
"""

from pathlib import Path
from datetime import timedelta
import os

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/3.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-c!k^37j-(5i0b2rm^d(luep%q%_6%girwc)q%o+y&c^&8+91v3'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'database',
    'main',
    'client',
```

```
    'farmer'
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'diplom.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            os.path.join('templates')
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'diplom.wsgi.application'

# Database
# https://docs.djangoproject.com/en/3.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Password validation
# https://docs.djangoproject.com/en/3.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
```

```
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityVal
idator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/3.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/3.2/howto/static-files/

STATIC_URL = '/static/'

# Default primary key field type
# https://docs.djangoproject.com/en/3.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
        'rest_framework.authentication.BasicAuthentication',
        'rest_framework.authentication.SessionAuthentication',
    ],
}

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=5),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
    'ROTATE_REFRESH_TOKENS': False,
    'BLACKLIST_AFTER_ROTATION': True,
    'UPDATE_LAST_LOGIN': False,
```

```
    'ALGORITHM': 'HS256',
    'SIGNING_KEY': SECRET_KEY,
    'VERIFYING_KEY': None,
    'AUDIENCE': None,
    'ISSUER': None,

    'AUTH_HEADER_TYPES': ('Bearer',),
    'AUTH_HEADER_NAME': 'HTTP_AUTHORIZATION',
    'USER_ID_FIELD': 'id',
    'USER_ID_CLAIM': 'user_id',
    'USER_AUTHENTICATION_RULE': 'rest_framework_simplejwt.authentication.default_us
er_authentication_rule',

    'AUTH_TOKEN_CLASSES': ('rest_framework_simplejwt.tokens.AccessToken',),
    'TOKEN_TYPE_CLAIM': 'token_type',

    'JTI_CLAIM': 'jti',

    'SLIDING_TOKEN_REFRESH_EXP_CLAIM': 'refresh_exp',
    'SLIDING_TOKEN_LIFETIME': timedelta(minutes=5),
    'SLIDING_TOKEN_REFRESH_LIFETIME': timedelta(days=1),
}
```

urls.py

```
"""diplom URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/3.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.conf.urls import url
from django.contrib import admin
from django.urls import path, include
from rest_framework_simplejwt.views import (
    TokenObtainPairView,
    TokenRefreshView,
    TokenVerifyView,
)
```

```
urlpatterns = [
    path('api-auth/', include('rest_framework.urls')),
    path('database/', include('database.urls')),
    path('', include('main.urls')),
    path('admin/', admin.site.urls),
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    path('api/token/verify/', TokenVerifyView.as_view(), name='token_verify'),
    path('client/', include('client.urls')),
    path('farmer/', include('farmer.urls'))
]
```

database

models.py

```
from django.db import models
from django.db.models.deletion import CASCADE
from django.contrib.auth.models import User

class PersonalData(models.Model):
    user = models.OneToOneField(User, on_delete= models.CASCADE)
    isFarmer = models.BooleanField(default= False)
    phoneNumber = models.CharField('user_phoneNumber', max_length = 13)
    money = models.DecimalField('balance', max_digits= 10, decimal_places= 2)

class Client(models.Model):
    personalData = models.OneToOneField(PersonalData, on_delete = models.CASCADE)
    addres = models.CharField('client_addres', max_length = 50)

class Farmer(models.Model):
    personalData = models.OneToOneField(PersonalData, on_delete = models.CASCADE)

class TypeOfPayment(models.Model):
    name = models.CharField('payment_type', max_length= 20)

class Payment(models.Model):
    client = models.ForeignKey(Client, on_delete= models.CASCADE)
    farmer = models.ForeignKey(Farmer, on_delete= models.CASCADE)
    typeOfPayment = models.ForeignKey(TypeOfPayment, on_delete=models.CASCADE)
    dateOfPayment = models.DateTimeField('payment_date', auto_now=False, auto_now_a
dd=False)
    sum = models.DecimalField('payment_sum', max_digits= 10, decimal_places= 3)

class DeliveryCompanies(models.Model):
    name = models.CharField('company_name', max_length= 40)
```

```
    price = models.DecimalField('company_priceForKM', max_digits= 10, decimal_place
s= 3)

class TypeOfDelivery(models.Model):
    name = models.CharField('typeOfDelivery_name', max_length= 30)

class Product(models.Model):
    name = models.CharField('product_name', max_length= 30)
    unitOfMeasure =  models.CharField('product_uom', max_length= 30)

class Storage(models.Model):
    farmer = models.ForeignKey(Farmer, on_delete= models.CASCADE)
    addres = models.CharField('storage_addres', max_length = 50)

class Stock(models.Model):
    product = models.OneToOneField(Product, on_delete= models.CASCADE)
    storage = models.ForeignKey(Storage, on_delete= models.CASCADE)
    amountOfProduct = models.DecimalField('stock_amountOfProduct', max_digits= 10,
decimal_places= 3)
    dateOfHarvest = models.DateTimeField('product_harvestDate', auto_now=False, aut
o_now_add=False)
    shelfLife = models.DateTimeField('product_shelfLife', auto_now=False, auto_now_
add=False)
    price = models.DecimalField('product_price', max_digits= 10, decimal_places= 3)

class Sale(models.Model):
    farmer = models.ForeignKey(Farmer, on_delete= models.CASCADE)
    product = models.ForeignKey(Product, on_delete= models.CASCADE)
    dateOfRequest = models.DateTimeField('sale_RequestDate', auto_now=False, auto_n
ow_add=False)
    dateOfHarvest = models.DateTimeField('sale_productHarvestTime', auto_now=False,
 auto_now_add=False)
    amount = models.DecimalField('sale_amountOfProduct', max_digits= 10, decimal_pl
aces= 3)
    price = models.DecimalField('sale_priceOfGrowingHarvest', max_digits= 10, decim
al_places= 3)

class Purchase(models.Model):
    client = models.ForeignKey(Client, on_delete= models.CASCADE)
    sale = models.ForeignKey(Sale, on_delete= models.CASCADE)
    dateOfRequest = models.DateTimeField('sale_RequestDate', auto_now=False, auto_n
ow_add=False)
    dateOfPurchase = models.DateTimeField('sale_RequestDate', auto_now=False, auto_
now_add=False)
    amount = models.DecimalField('purchase_amount', max_digits= 10, decimal_places=
 3)

class Trade(models.Model):
    client = models.ForeignKey(Client, on_delete= models.CASCADE)
    farmer = models.ForeignKey(Farmer, on_delete= models.CASCADE)
    product = models.ForeignKey(Product, on_delete= models.CASCADE)
```

```
    purchase = models.ForeignKey(Purchase, on_delete= models.CASCADE)
    typeOfPayment = models.ForeignKey(TypeOfPayment, on_delete= models.CASCADE)
    payment = models.ForeignKey(Payment, on_delete= models.CASCADE)
    typeOfDelivery = models.OneToOneField(TypeOfDelivery, on_delete= models.CASCADE
)
    deliveryCompany = models.ForeignKey(DeliveryCompanies, on_delete= models.CASCAD
E)
    price = models.DecimalField('trade_priceOfProduct', max_digits= 10, decimal_pla
ces= 3)
    amount = models.DecimalField('trade_amountOfProduct', max_digits= 10, decimal_p
laces= 3)
    distance = models.DecimalField('trade_distanceOfDelivery', max_digits= 10, deci
mal_places= 3)
```

client

serializers.py

```
from database.serializers import UserSerializer
import client
from django.db.models import fields
from database.models import *
from rest_framework import serializers
from django.contrib.auth.models import User



class FarmerPersonalDataSerializer(serializers.ModelSerializer):
    user = UserSerializer()
    class Meta:
        model = PersonalData
        fields = '__all__'

class ProductSerializer(serializers.ModelSerializer):
    class Meta:
        model = Product
        fields = '__all__'

class FarmerSerializer(serializers.ModelSerializer):
    personalData = FarmerPersonalDataSerializer()
    class Meta:
        model = Farmer
        fields = '__all__'

class ClientSerializer(serializers.ModelSerializer):
    class Meta:
```

```
        model = Client
        exclude = ['personalData']

class PurchaseSerializer(serializers.ModelSerializer):
    class Meta:
        model = Purchase
        fields = '__all__'

class TypeOfPaymentSerializer(serializers.ModelSerializer):
    class Meta:
        model = TypeOfPayment
        fields = '__all__'

class PaymentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Payment
        fields = '__all__'

class TypeOfDeliverySerializer(serializers.ModelSerializer):
    class Meta:
        model = TypeOfDelivery
        fields = '__all__'

class DeliveryCompanySerializer(serializers.ModelSerializer):
    class Meta:
        model = DeliveryCompanies
        fields = '__all__'

class PersonalDataSerializer(serializers.ModelSerializer):
    client = ClientSerializer()
    class Meta:
        model = PersonalData
        exclude = ['user']

class ClientProfileSerializer(serializers.ModelSerializer):
    personaldata = PersonalDataSerializer()

    class Meta:
        model = User
        fields = '__all__'

class SaleSerializer(serializers.ModelSerializer):
    farmer = FarmerSerializer()
    product = ProductSerializer()
    class Meta:
        model = Sale
        fields = '__all__'



class ClientBuySerializer(serializers.ModelSerializer):
```

```
    sale = SaleSerializer()
    class Meta:
        model = Purchase
        fields = '__all__'


class ClientTradeSerializer(serializers.ModelSerializer):
    product = ProductSerializer()
    client = ClientSerializer()
    farmer = FarmerSerializer()
    purchase = PurchaseSerializer()
    typeOfPayment = TypeOfPaymentSerializer(many = False)
    payment = PaymentSerializer()
    typeOfDelivery = TypeOfDeliverySerializer()
    deliveryCompany = DeliveryCompanySerializer()
    class Meta:
        model = Trade
        fields = '__all__'

class ClientDeliverySerializer(serializers.ModelSerializer):
    trade = ClientTradeSerializer()
    class Meta:
        model = TypeOfDelivery
        fields = '__all__'


class ClientBinSerializer(serializers.ModelSerializer):

    client = ClientSerializer()
    sale = SaleSerializer()
    class Meta:
        model = Purchase
        fields = '__all__'

class ClientPaySerializer(serializers.ModelSerializer):
    client = ClientSerializer()
    sale = SaleSerializer()
    class Meta:
        model = Purchase
        fields = '__all__'
```

urls.py

```
from django.urls import path

from . import views

urlpatterns = [
    path('menu/', views.ProductListAPI.as_view()),
```

```
    path('profile/', views.ClientProfileAPI.as_view()),
    path('profile/pay/', views.ClientPayAPI.as_view()),
    path('history/', views.ClientHistoryAPI.as_view()),
    path('bin/', views.ClientBinAPI.as_view()),
    path('buy/', views.ClientBuyAPI.as_view()),
    path('buy/delivery', views.TypeOfDeliveryAPI.as_view())

]
```

views.py

```
from database.models import PersonalData
from django.http import HttpResponse
from rest_framework import viewsets, permissions, generics
from rest_framework.response import Response
from .serializers import *

class ProductListAPI(generics.GenericAPIView):

    serializer_class = SaleSerializer
    queryset = Sale.objects.all()

    def get(self, request):
        return Response(self.get_serializer(self.get_queryset(), many = True).data,
 status = 200)

    def post(self, request):

        return

class ClientProfileAPI(generics.GenericAPIView):

    serializer_class = ClientProfileSerializer
    queryset = User.objects.all()

    def get(self, request):

        return Response(self.get_serializer(self.get_queryset(), many = True).data,
 status = 200)


    def post(self, request):

        return


class ClientPayAPI(generics.GenericAPIView):

    serializer_class = ClientPaySerializer
```

```
    queryset = Purchase.objects.all()

    def get(self, request):

        return Response(self.get_serializer(self.get_queryset(), many = True).data,
 status = 200)


    def post(self, request):

        return

class ClientHistoryAPI(generics.GenericAPIView):

    serializer_class = ClientTradeSerializer
    queryset = Trade.objects.all()

    def get(self, request):

        return Response(self.get_serializer(self.get_queryset(), many = True).data,
 status = 200)

    def post(self, request):

        return

class ClientBinAPI(generics.GenericAPIView):

    serializer_class = ClientBinSerializer
    queryset = Purchase.objects.all()

    def get(self, request):

        return Response(self.get_serializer(self.get_queryset(), many = True).data,
 status = 200)

    def post(self, request):

        return

class ClientBuyAPI(generics.GenericAPIView): #

    serializer_class = ClientBuySerializer
    queryset = Purchase.objects.all()

    def get(self, request):

        return Response(self.get_serializer(self.get_queryset(), many = True).data,
 status = 200)

    def post(self, request):
```

```
        return


class TypeOfDeliveryAPI(generics.GenericAPIView):

    serializer_class = ClientDeliverySerializer
    queryset = TypeOfDelivery.objects.all()

    def get(self, request):

        return Response(self.get_serializer(self.get_queryset(), many = True).data,
 status = 200)

    def post(self, request):

        return
```

farmer

serializers.py

```
from django.db.models import fields
from database.models import *
from rest_framework import serializers
from django.contrib.auth.models import User

class FarmerSerializer(serializers.ModelSerializer):
    class Meta:
        model = Farmer
        exclude = ['personalData']

class FarmerPersonalDataStorageSerializer(serializers.ModelSerializer):
    class Meta:
        model = PersonalData
        fields = '__all__'

class FarmerStorageSerializer(serializers.ModelSerializer):
    personalData = FarmerPersonalDataStorageSerializer()
    class Meta:
        model = Farmer
        fields = '__all__'

class StorageSerializer(serializers.ModelSerializer):
    farmer = FarmerStorageSerializer()
    class Meta:
        model = Storage
        fields = '__all__'

class StockSerializer(serializers.ModelSerializer):
    storage = StorageSerializer()
```

```
    class Meta:
        model = Stock
        fields = '__all__'

class ProductSerializer(serializers.ModelSerializer):
    stock = StockSerializer()
    class Meta:
        model = Product
        fields = '__all__'


class ClientSerializer(serializers.ModelSerializer):
    class Meta:
        model = Client
        fields = '__all__'

class PurchaseSerializer(serializers.ModelSerializer):
    class Meta:
        model = Purchase
        fields = '__all__'

class TypeOfPaymentSerializer(serializers.ModelSerializer):
    class Meta:
        model = TypeOfPayment
        fields = '__all__'

class PaymentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Payment
        fields = '__all__'

class TypeOfDeliverySerializer(serializers.ModelSerializer):
    class Meta:
        model = TypeOfDelivery
        fields = '__all__'

class DeliveryCompanySerializer(serializers.ModelSerializer):
    class Meta:
        model = DeliveryCompanies
        fields = '__all__'


class PersonalDataSerializer(serializers.ModelSerializer):
    farmer = FarmerSerializer()
    class Meta:
        model = PersonalData
        exclude = ['user']
```

```
class FarmerProfileSerializer(serializers.ModelSerializer):
    personaldata = PersonalDataSerializer()

    class Meta:
        model = User
        fields = '__all__'

class FermerPaySerializer(serializers.ModelSerializer):
    class Meta:
        model = PersonalData
        fields = '__all__'




class FarmerHistorySerializer(serializers.ModelSerializer):
    product = ProductSerializer()
    client = ClientSerializer()
    farmer = FarmerSerializer()
    purchase = PurchaseSerializer()
    typeOfPayment = TypeOfPaymentSerializer(many = False)
    payment = PaymentSerializer()
    typeOfDelivery = TypeOfDeliverySerializer()
    deliveryCompany = DeliveryCompanySerializer()
    class Meta:
        model = Trade
        fields = '__all__'

class ProductAddSerializer(serializers.ModelSerializer):
    stock = StockSerializer()
    class Meta:
        model = Product
        fields = '__all__'

class ProductEditSerializer(serializers.ModelSerializer):
    stock = StockSerializer()
    class Meta:
        model = Product
        fields = '__all__'
```

urls.py

```
from django.urls import path

from . import views

urlpatterns = [
```

```
    path('profile/', views.FarmerProfileAPI.as_view()),
    path('profile/pay/', views.FarmerPayAPI.as_view()),
    path('history/', views.FarmerHistoryAPI.as_view()),
    path('storages/', views.StorageAPI.as_view()),
    path('storages/edit/', views.StorageEditAPI.as_view()),
    path('products/', views.ProductListAPI.as_view()),
    path('products/add/', views.ProductAddAPI.as_view()),
    path('products/edit/', views.ProductEditAPI.as_view())

]
```

views.py

```
from rest_framework.response import Response
from .serializers import *
from django.http import HttpResponse
from rest_framework import generics

class FarmerProfileAPI(generics.GenericAPIView):

    serializer_class = FarmerProfileSerializer
    queryset = User.objects.all()

    def get(self, request):

        return Response(self.get_serializer(self.get_queryset(), many = True).data,
 status = 200)


    def post(self, request):

        return

class FarmerPayAPI(generics.GenericAPIView):
    serializer_class = FermerPaySerializer
    queryset = PersonalData.objects.all()

    def get(self, request):

        return Response(self.get_serializer(self.get_queryset(), many = True).data,
 status = 200)

    def post(self, request):

        return


class FarmerHistoryAPI(generics.GenericAPIView):
```

```
    serializer_class = FarmerHistorySerializer
    queryset = Trade.objects.all()

    def get(self, request):

        return Response(self.get_serializer(self.get_queryset(), many = True).data,
 status = 200)

    def post(self, request):

        return


class StorageAPI(generics.GenericAPIView):

    serializer_class = StorageSerializer
    queryset = Storage.objects.all()

    def get(self, request):

        return Response(self.get_serializer(self.get_queryset(), many = True).data,
 status = 200)

    def post(self, request):

        return



class StorageEditAPI(generics.GenericAPIView):

    serializer_class = StorageSerializer
    queryset = Storage.objects.all()

    def get(self, request):

        return Response(self.get_serializer(self.get_queryset(), many = True).data,
 status = 200)

    def post(self, request):

        return


class ProductListAPI(generics.GenericAPIView):

    serializer_class = ProductSerializer
    queryset = Product.objects.all()

    def get(self, request):
```

```
        return Response(self.get_serializer(self.get_queryset(), many = True).data,
 status = 200)

    def post(self, request):

        return



class ProductAddAPI(generics.GenericAPIView):

    serializer_class = ProductAddSerializer
    queryset = Product.objects.all()

    def get(self, request):

        return Response(self.get_serializer(self.get_queryset(), many = True).data,
 status = 200)

    def post(self, request):

        return



class ProductEditAPI(generics.GenericAPIView):

    serializer_class = ProductEditSerializer
    queryset = Product.objects.all()

    def get(self, request):

        return Response(self.get_serializer(self.get_queryset(), many = True).data,
 status = 200)

    def post(self, request):

        return
```

ДОДАТОК Б

ВІДГУК

керівника економічного розділу

на кваліфікаційну роботу бакалавра

на тему:

«Розробка веб-додатка товарно-інформаційної системи на мові Python та фреймворку Django»

студента групи 121-17-1 Бізюков Владислава Євенійовича

ДОДАТОК В

Перелік файлів на диску

Ім’я файлу	Опис
Пояснювальні документи	
Кваліфікаційна робота Бізюков.docx	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Кваліфікаційна робота Бізюков.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF
Програма	
Бізюков.rar	Архів. Містить коди програми
Презентація	
Бізюков.ppt	Презентація кваліфікаційної роботи