

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПОШУКУ СЕЗОННИХ ПРОДУКТІВ
SEASONAL PRODUCT SEARCH SOFTWARE

Виконав: студент(ка) 4 курсу, групи КНТ-129сп
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Логвиненко Д.С.

(прізвище та ініціали)

Керівник Скрупський С.Ю.

(прізвище та ініціали)

Рецензент Голуб Т.В.

(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Логвиненка Даниїла Сергійовича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення пошуку сезонних продуктів
Seasonal Product Search Software

керівник проєкту (роботи) Скрупський Степан Юрійович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «27» квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) червень 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Скрупський С.Ю., доцент		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання «4» квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка)

(підпис)

Логвиненко Д.С.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Скрупський С.Ю.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 82 с., 21 рис., 2 табл., 3 дод., 10 джерел.

СЕЗОННИЙ ПРОДУКТ, СОРТ ПРОДУКТУ, ФЕРМЕР, ПОКУПЕЦЬ, ПЕРЕДЗАМОВЛЕННЯ.

Об'єкт роботи – процес продажу сезонних продуктів. Предмет роботи – програмне забезпечення пошуку сезонних продуктів. Мета роботи – розробити програмне забезпечення пошуку сезонних продуктів для підтримки прямої взаємодії між фермерами і покупцями.

Матеріали, методи та технічні засоби: мова Python, серверний вебфреймворк Django, система керування базами даних MySQL.

Результати. Проблему спочатку було проаналізовано, на основі чого сформовано функціональні вимоги, побудовано схему бази даних, за якою розроблено моделі, структуровано проєкт з визначенням вебсторінок з відповідними шаблонами і представленнями, і реалізовано програму в цілому.

Висновки. Розроблена програма дозволяє створити майданчик для прямої взаємодії між фермерами і покупцями, які знаходяться в пошуку сезонних продуктів, для уникнення третіх сторін і отримання якісних свіжих продуктів. При реалізації використано ідею, що продукти можуть пересилатися сервісами доставки, а можуть надаватися прямо на фермі покупцю, якщо він готовий самостійно дістатися її. Окрім того за необхідності фермери можуть самостійно розподіляти наявну кількість продуктів між покупцями за допомогою програми.

Галузь використання. Ринок продажу сезонних продуктів в ракурсі організації платформи між фермерами і покупцями.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 82 pages, 21 figures, 2 tables, 3 appendixes, 10 sources.

SEASONAL PRODUCT, PRODUCT SORT, FARMER, BUYER, PRE-ORDER.

The object of work is the process of selling seasonal products. The subject of work is a seasonal product search software. The purpose of the work is to develop software for finding seasonal products to support direct interaction between producers of seasonal products and buyers.

Materials, methods and technical means: Python language, Django server web framework, MySQL database management system.

Results. The problem was first analyzed. Then functional requirements were stated based on the analysis results. A database schema was built. According to the schema software models were developed, a project was structured with the definition of web pages with relevant templates and views, and the program as a whole was implemented.

Conclusions. The developed program allows to create a platform for direct interaction between farmers and buyers who search for seasonal products, to avoid third parties and to get quality fresh products. The implemented idea presumes that the products could be sent by delivery services and could be provided directly on the farm to the buyer, if he is ready to get it himself. In addition, if necessary, farmers can independently distribute the available amount of products among customers through the program.

Field of use. The market for the sale of seasonal products in the perspective of the organization of the platform between the farmers and buyers.

ЗМІСТ

	С.
Перелік скорочень та умовних познач	7
Вступ	8
1 Аналіз предметної області	9
1.1 Переваги і важливість продажу сезонних продуктів	9
1.2 Огляд програмних аналогів	10
1.3 Порівняння програмних аналогів	12
1.4 Висновки за розділом 1	13
2 Розробка архітектури програми	15
2.1 Визначення функціональних вимог до програми	15
2.2 Вибір мови програмування	17
2.3 Проектування бази даних	18
2.4 Висновки за розділом 2	28
3 Розробка програми	29
3.1 Алгоритм функціонування програми	29
3.2 Опис реалізованих класів програми	31
3.3 Висновки за розділом 3	40
4 Експлуатація та тестування програми	41
4.1 Призначення програми	41
4.2 Умови виконання програми	41
4.3 Експлуатація програми	41
4.4 Тестування програми	45
4.5 Висновки за розділом 4	45
Висновки	46
Перелік джерел посилання	47
Додаток А Технічне завдання	48
Додаток Б Текст програми	52
Додаток В Слайди презентації	76

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;

СКБД – система керування базами даних.

ВСТУП

Продукти, які вирощуються та споживаються протягом відповідних сезонів, є більш насиченими поживними речовинами: коли продукти вирощують не в сезон, вони не можуть дотримуватися свого природного ритму росту та дозрівання, а для того, щоб певні фрукти та овочі були доступними цілий рік, використовуються післязбиральні обробки (хімічні речовини, гази та теплові процеси), деякі продукти також покриті їстівною плівкою для їх захисту, ці процеси дозволяють виробляти продукти у великих кількостях за рахунок уповільнення процесу дозрівання, вони також допомагають захистити продукцію від бактерій та інших хвороботворних мікроорганізмів під час довгої подорожі від полів до місцевого продуктового магазину [1].

При цьому продукти, вирощені більш традиційним способом, зібрані самими фермерами, можуть мати кращі смакові якості, бути більш поживними на вітаміни. Для людей, які віддають перевагу здоровому харчуванню, це важливо. Окрім того придбання сезонних продуктів від місцевих фермерів є певною традицією зокрема в нашій країні, які з часом була дещо загублена в певному сенсі. Цьому сприяло зокрема те, що відсутні прямі зв'язки між фермерами і покупцями, адже фермерам зазвичай незручно долати відстань до міста, тому прямі контакти порушуються. Актуальним тому є налагодження таких контактів. З цією метою і розробляється програмне забезпечення в цій роботі, присвячене пошуку сезонних продуктів.

У результаті мета роботи була визначена як розробити програмне забезпечення пошуку сезонних продуктів для підтримки прямої взаємодії між фермерами і покупцями.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Переваги і важливість продажу сезонних продуктів

Перевагами сезонних продуктів є:

- корисність: хоча на даний момент фермери можуть задовольняти споживчий попит цілий рік, дослідники виявили, що штучно дозрілі продукти часто не такі поживні чи смачні, як дозрілі природним шляхом, дослідження також показали, що сезонність може впливати на поживність інших харчових продуктів, зокрема, молочних продуктів;

- смакові якості: продукти масового виробництва, призначені для задоволення світового споживчого попиту, як правило, страждають від відсутності смаку, метою великих комерційних господарств є виробництво такого обсягу продукту для задоволення високого попиту, при цьому кількість і зовнішній вигляд переважають смак, селективне розведення сприяє рівномірному дозріванню та переваженню зберігання над смаком, що призводить до тьмяних помідорів і несмачної полуниці, з іншого боку, природно дозрілі фрукти та овочі, вирощені та зібрані в сезон, зазвичай сповнені смаку та поживних речовин;

- корисність для навколишнього середовища: фрукти, овочі, які куплені в продуктовому магазині, можуть бути поставлені від місцевого фермера, а можуть бути поставлені з іншого місця в країні або взагалі прибути літаком з іншого континенту, вплив такої далекої подорожі відчутний не лише на харчовій цінності їжі та вартості, але й на навколишньому середовищі, включаючи викиди;

- вартість: фрукти чи овочі в сезон відзначаються не тільки великою кількістю, але і через це нижчою ціною, відповідно в несезон за ці фрукти і овочі доводиться платити значно більше [1]-[2].

Окрім всього цього можна також окремо вказати, чому важливо їсти сезонну, фактично місцеву їжу:

- купівля місцевих продуктів харчування допомагає підтримувати місцеві ферми та підтримувати сільськогосподарські угіддя в громаді;
- місцева їжа підтримує місцеву економіку: гроші, які витрачаються на продукти місцевих фермерів і виробників, залишаються в громаді та реінвестуються в інші місцеві підприємства, крім того, їжа, вирощена на місцевому рівні, оброблена та розповсюджена на місцевому рівні (наприклад, у місцеві ресторани), створює робочі місця і згодом допомагає стимулювати місцеву економіку;
- місцеві виробники можуть розповісти, як вирощували продукти: якщо покупка здійснюється у фермерів, є можливість запитати, які методи вони використовують для вирощування та збирання врожаю [3].

Зважаючи на все це, варто вказати на актуальність розробки, яку запропоновано виконати в роботі. Розроблена програма має дозволити безпосередньо взаємодіяти фермерам і покупцям. Фермерами фактично можуть бути малі бізнеси, аж до того, що ці фрукти або овочі вирощуються на невеликому городі, на даху або на балконі в місті. У свою чергу покупцями можуть бути окремі люди, що живуть недалеко, можуть бути також місцеві ресторани. Програма тоді дозволить їм, по-перше, дізнатися один про одного, а по-друге, замовити продукти, коли вони потрібні. Це основна ідея розробки.

1.2 Огляд програмних аналогів

Вебсайт Seasonalfoodguide.org [3] (рис. 1.1) є путівником в світ сезонних продуктів. Окрім вебсайту доступні також застосунки для мобільних пристроїв. Це зручний інструмент для пошуку розповсюдження сезонних продуктів у певних місцинах. Для виконання пошуку потрібно спочатку обрати штат США, потім задати частину місяця зі списку, а тоді обрати потрібний продукт або відобразити всі розповсюджені в цій місцині в

цей період продукти. За кожним продуктом можна отримати інформацію про нього, його корисність.

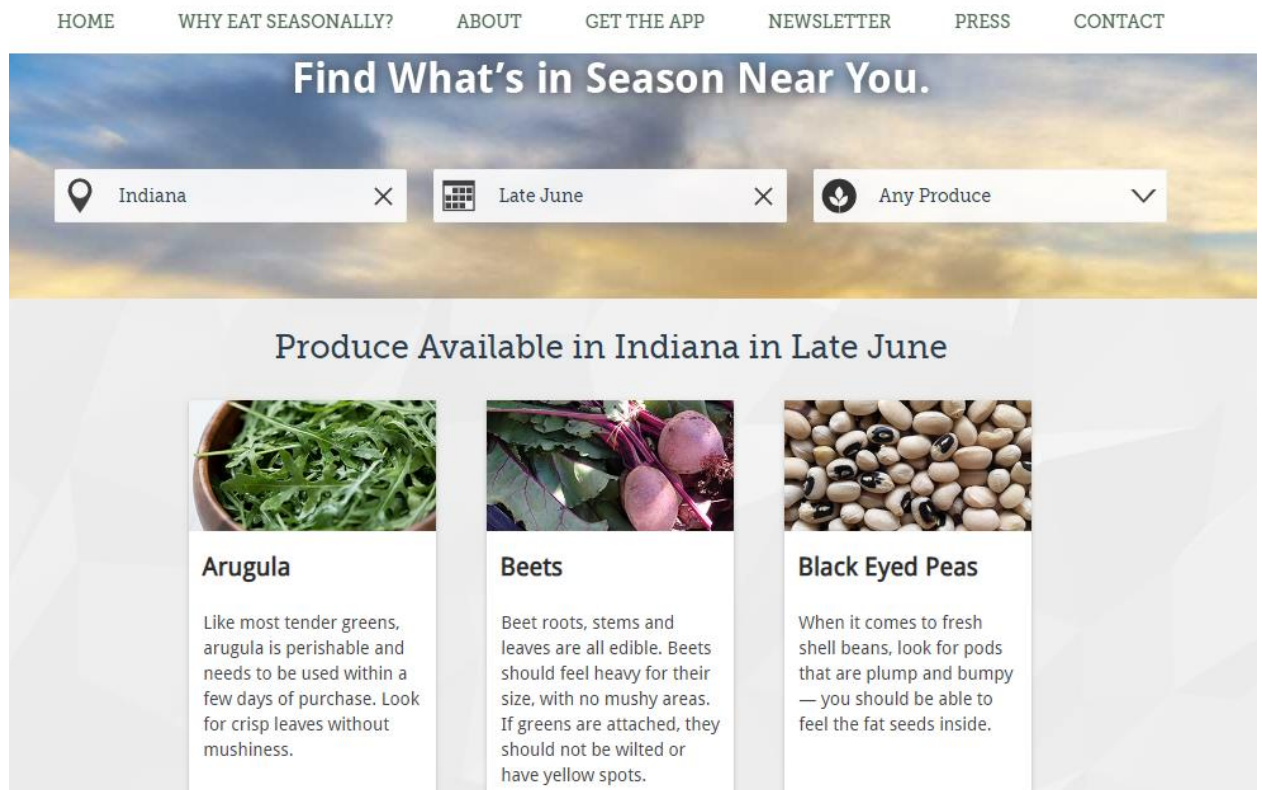


Рисунок 1.1 – Вебсайт Seasonalfoodguide.org

Вебсайт Sharingfarm.ca [4] (рис. 1.2) дозволяє отримати інформацію про сезонні продукти на фермі в Канаді. Звичайно, що це один з прикладів таких інструментів. Багато різних ферм у світі мають свої власні вебсайти. Фактично в даному випадку надається можливість отримати інформацію про ферму, замовити кошик з продуктами поточного сезону з магазину вебсайту або профінансувати такий кошик для людей, які цього собі дозволити не можуть. Не завжди можливість хоч якого-небудь замовлення є на вебсайті ферм, а у даному випадку, на сайті цієї ферми, у поточний момент магазин не працював.

604-227-6210
info@sharingfarm.ca
f
t
r
0 Items

Home
About
Online Store
Sponsorship Opportunities
Volunteering
Harvest Basket
Farm Updates
Contact Us

Fresh Vegetables Straight From the Farm

Support our charitable work at The Farm by enjoying weekly seasonal harvests of fresh sustainably grown vegetables. Sign up now for a Harvest Basket in 2022.

- 17 weeks, from mid-June through to October (dates TBC), of delicious produce
- The cost is \$37/week, for a total of \$625, payable at the time of registration.
- Enough produce to feed a family of four. Shares may be split with a friend, either splitting the box each week, or alternating pick up weeks. (If you can't find someone to share with you email us at:



DONATE NOW

Your donation will change the notion of "Food Bank Food". Fresh produce for families in need.

Tweets by @sharingfarm

Рисунок 1.2 – Вебсайт Sharingfarm.ca

1.3 Порівняння програмних аналогів

Програм, які стосуються сезонних продуктів саме відносно тієї точки, з якої це розглядається в роботі, виявлено не було. Проте було виконано порівняння (табл. 1.1) з тими програмами, які загалом сфери вживання сезонних продуктів напряду стосуються.

Жодна з цих програм не надає можливість замовити сезонні продукти. Це є особливістю власної розробки.

Sharingfarm.ca стосується тільки однієї конкретної ферми, а працювати різним фермерам, створюючи таким чином майданчик для них, програма не надає можливість.

Seasonalfoodguide.org популяризує сезонні продукти, дозволяє дізнаватися про них в різних місцях США, але вона знову ж таки не дозволяє контактувати з фермерами, це тільки початковий інструмент, після якого потрібно далі виконувати пошук, де такі продукти можна купити.

Таблиця 1.1 – Порівняння програмних аналогів

Показник	Seasonalfoodguide.org	Sharingfarm.ca	Власна розробка
Замовлення сезонних продуктів	Ні	Так	Так
Інформація про сезонні продукти на фермі	Ні	Так	Так
Інформація про сезонні продукти	Так	Так	Так
Доступ різних фермерів	Ні	Ні	Так

Таким чином, розробка, яка виконується в роботі, повинна дозволити фермеру знаходити нових покупців за рахунок наявності великого торгового простору, де різні покупці шукають різні продукти і їх стає більше за рахунок наявності різних пропозицій, окрім того використовувати дані платформи замість наявності окремих соціальних мереж, де в ручному режимі обробляються замовлення, що є більш часто використовуваним рішенням навіть порівняно з вебсайтами окремих ферм, які часто не мають і інструментів замовлення. Фактично фермер не вимушений їхати розвозити власні продукти або виїжджати на фермерський ринок і втрачати на це власний час.

1.4 Висновки за розділом 1

Проведений огляд проблеми виявив переваги сезонних продуктів, а також важливість купівлі саме сезонних продуктів у невеликих фермерів порівняно з масово вирощуваними продуктами. Проведений аналіз аналогів

вказав на відсутність подібного інструменту, що об'єднував би різних фермерів у одній програмі з різними покупцями.

Завдання роботи:

- дослідження проблеми продажу сезонних продуктів;
- визначення засобів розробки;
- витягання вимог до програми;
- проєктування програми пошуку сезонних продуктів;
- реалізація програми.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Визначення функціональних вимог до програми

Загальну проблему продажу сезонних продуктів, яку було проаналізовано в першому розділі, було доведено до набору функціональних вимог, який перетворено на набір сценаріїв використання програми (рис. 2.1).

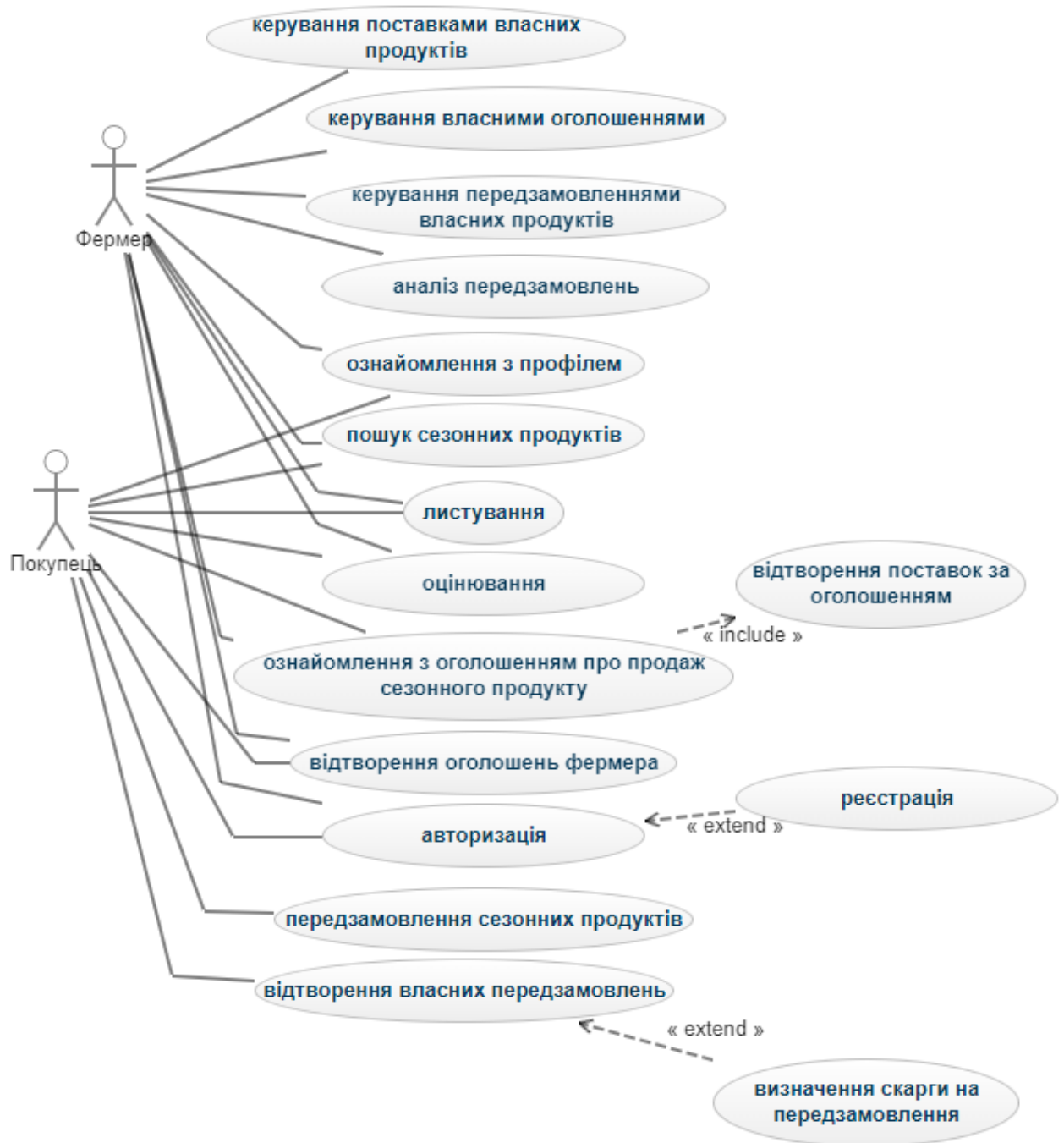


Рисунок 2.1 – Діаграма прецедентів

Відповідні функціональні вимоги:

- пошук оголошень про продаж потрібних сезонних продуктів;
- відтворення власних оголошень;
- визначення оголошення;
- оновлення оголошення;
- передзамовлення сезонних продуктів;
- керування передзамовленням сезонних продуктів;
- оцінювання покупця;
- оцінювання сезонних продуктів за оголошенням;
- ознайомлення з оголошенням про продаж сезонного продукту;
- відтворення оголошень фермера;
- виставлення нової поставки продуктів за оголошенням;
- оновлення поставки;
- ознайомлення з поставкою продуктів за оголошенням;
- відтворення власних поставок;
- відтворення поставок за оголошенням;
- оновлення передзамовлення;
- відтворення передзамовлень, адресованих фермеру;
- відтворення власних передзамовлень;
- пошук потрібних сезонних продуктів;
- аналіз кількості передзамовлень у фермера за місцем;
- відтворення списку власних оцінювань оголошень;
- відтворення списку оцінювань оголошень фермера;
- аналіз обсягів замовлень сортів сезонних продуктів у фермера;
- авторизація, реєстрація;
- відтворення списку власних оцінювань покупців;
- відтворення списку оцінювань покупця;
- надсилення листа;
- визначення скарги на передзамовлення;
- відтворення списку листів між користувачами;

- відтворення списку продавців за місцем розташування;
- ознайомлення з профілем покупця;
- ознайомлення з профілем фермера.

2.2 Вибір мови програмування

Визначені функціональні вимоги до програми продемонстрували, що з нею будуть працювати різні користувачі, при цьому це можуть бути і користувачі з різними ролями, до того ж кожен з них буде працювати з власного пристрою віддалено. Для такого типу роботи придатними є застосунки для мобільних пристроїв і вебсистеми. У першому випадку проте потрібно розробити окремі застосунки для фермерів і покупців, а до того ж кожен з цих застосунків має передбачати окремі версії під різні версії операційних систем, до того ж таких популярних мобільних систем дві. Тому для спрощення розробки було вирішено створювати вебсистему.

Розробку програми доцільно виконувати зокрема достатньо застосовуваними в цій сфері мовами програмування – Python [5]-[6] або Java [7] (табл. 2.1).

Таблиця 2.1 – Вибір мови програмування для розробки програми

Показник	Java	Python
Менший обсяг коду	Ні	Так
Менші витрати часу на написання коду	Ні	Так
Система типізації динамічна	Ні	Так
Менший час виконання	Так	Ні
Реалізація вебсистем	Так	Так

Тому було виконано порівняння цих мов як засобів розробки, а в підсумку обрано Python, який поступається щодо обраних критеріїв тільки за рахунок більшого часу виконання, проте на практиці в таких системах за використання потужних серверів навіть за великої кількості запитів цієї мови достатньо для створення потужних вебсистем.

2.3 Проєктування бази даних

Проєктування бази даних (БД) було одним з основних моментів, що визначали процес розробки програми, оскільки виділені тут сутності впливали на реалізацію програми безпосередньо, зокрема на реалізацію функцій програми, на її структурування, тому розробці БД було приділено особливу увагу. Розробка виконувалась з застосуванням системи керування БД (СКБД) MySQL. Під час проєктування використання цієї СКБД безпосередньо впливало на ті типи полів таблиць, які були використані.

Виділені у процесі проєктування сутності БД:

- користувач – обліковий запис користувача в системі;
- фермер – людина або люди, які продають сезонні продукти, при цьому передбачається, що вони їх самі і виробляють: формулювання виробляють використовується в загальному вигляді для позначення того, що певні продукти також можуть якось оброблятися, а не тільки вирощуватися, але загалом можна застосовувати і термін вирощують (сезонні продукти) замість виробляють, до того ж варто відзначити, що фермери є користувачами, які працюють з програмою заради того, щоб продати свої сезонні продукти покупцям;
- покупець – користувач, який бере участь в роботі програми для покупки сезонних продуктів від самих фермерів;
- сезонний продукт – основна сутність предметної області, підтримку продажу яких свіжими від фермерів безпосередньо покупцям без участі посередників і організовує програма;

– сорт сезонного продукту – конкретний сорт сезонного продукту, який продається фермером: використовується для більш чіткого розуміння, що саме пропонується на продаж;

– оголошення – сутність, яка визначає увідомлення про те, що певний фермер вирощує і продає певний сорт сезонного продукту;

– місце знаходження – місто, населений пункт або селище, де знаходиться саме виробництво (наприклад, ферма) або відділення сервісу доставки;

– область – територіальна одиниця, які групують місця знаходження: в Україні це області;

– поставка – виробництво (збирання) сезонного продукту певного сорту за відповідним оголошенням на певну дату, що визначає, яка саме кількість продукту готова (наприклад, зібрана) на певний час;

– передзамовлення – передзамовлення певної поставки сезонних продуктів покупцем, якщо фермер допускає таке передзамовлення;

– сервіс доставки – служба, яка реалізує доставку між містами і місцями знаходження ферми і покупця;

– відділення – пункт відправлення і отримання посилки;

– оцінювання оголошення – оцінювання сезонних продуктів фермера покупцем;

– оцінювання покупця – оцінювання покупця фермером, який продав йому продукти;

– лист – повідомлення, яке за необхідності надсилає покупець фермеру, а фермер покупцю.

Таблиця облікового запису користувача User надає характеристики відповідної сутності за полями:

– id – ідентифікатор облікового запису;

– username – логін облікового запису користувача;

– first_name – ім'я;

– last_name – прізвище;

- email – електронна пошта користувача;
- password – пароль облікового запису користувача;
- is_staff – доступ до адміністративної панелі;
- is_active – чи є обліковий запис активним;
- is_superuser – доступ до всіх можливостей;
- last_login – остання авторизація облікового запису користувача;
- date_joined – дата додавання облікового запису.

Таблиця фермера Farmer надає характеристики відповідної сутності за полями:

- user_id – ідентифікатор облікового запису користувача;
- department_id – ідентифікатор основного відділення служби доставки, через яке працює фермер;
- address – адреса ферми;
- problems – кількість проблемних відправлень у фермера з його вини;
- phone – номер телефону фермера.

Зв'язком, який з'єднує таблиці User і Farmer виступає зв'язок один до одного: фермер може мати тільки один обліковий запис, запис у свою чергу використовується лише одним фермером або жодним.

Зв'язком, який з'єднує таблиці Department і Farmer виступає зв'язок один до багатьох: відділення сервісу доставки може працювати з різними фермерами, фермер у свою чергу використовує лише одне відділення як основне для відправки (використовується по суті як географічний орієнтир, додатково до адреси, за якою розташовано виробництво).

Таблиця покупця Buyer надає характеристики відповідної сутності за полями:

- user_id – ідентифікатор облікового запису користувача;
- mark – середня оцінка покупця;
- problems – кількість проблемних отримань у покупця з його вини;
- phone – номер телефону покупця.

Зв'язком, який з'єднує таблиці User і Buyer виступає зв'язок один до одного: покупець може мати тільки один обліковий запис, запис у свою чергу використовується лише одним покупцем або жодним.

Таблиця сезонного продукту Product надає характеристики відповідної сутності за полями:

- id – ідентифікатор сезонного продукту;
- prodgroup – група продуктів, до якої належить цей сезонний продукт;
- name – назва сезонного продукту;
- mainseason – основний сезон року, під час якого активно отримується цей сезонний продукт;
- pic – шлях до файлу з фотографією сезонного продукту;
- features – опис особливостей сезонного продукту, зокрема його корисності.

Таблиця сорту сезонного продукту Productsort надає характеристики відповідної сутності за полями:

- id – ідентифікатор сорту сезонного продукту;
- product_id – ідентифікатор сезонного продукту, до якого належить цей сорт;
- name – назва сорту сезонного продукту;
- features – опис сорту сезонного продукту, зокрема особливостей його корисності.

Зв'язком, який з'єднує таблиці Product і Productsort виступає зв'язок один до багатьох: сорт сезонного продукту відноситься тільки до одного сезонного продукту, а сам сезонний продукт може мати різні сорти.

Таблиця оголошення про продаж сезонного продукту Advertisement надає характеристики відповідної сутності за полями:

- id – ідентифікатор оголошення;
- farmer_id – ідентифікатор фермера;

- productsort_id – ідентифікатор сорту сезонного продукту, який продається за цим оголошенням;
- area_id – район, в якому виробляється (вирощується) цей сезонний продукт;
- mark – середня оцінка за оголошенням;
- enabled – доступний продаж за оголошенням в поточний момент або ні;
- started_at – дата створення оголошення;
- season_start – початок сезону;
- season_end – завершення сезону;
- pic – шлях до файлу з фотографією сезонних продуктів за цим оголошенням.

Зв'язком, який з'єднує таблиці Farmer і Advertisement виступає зв'язок один до багатьох: оголошення може бути сформовано тільки одним фермером, фермер у свою чергу може мати різну кількість оголошень.

Зв'язком, який з'єднує таблиці Area і Advertisement виступає зв'язок один до багатьох: оголошення може стосуватися тільки однієї області, де виробляється (вирощується) сезонний продукт, область у свою чергу може мати різну кількість оголошень.

Зв'язком, який з'єднує таблиці Productsort і Advertisement виступає зв'язок один до багатьох: оголошення може представляти тільки один сорт одного сезонного продукту (якщо потрібно продати різні сорти, різні сезонні продукти, то потрібно створювати різні оголошення), сорт сезонного продукту може бути задіяний в різних оголошеннях, тобто фактично продаватися різними фермерами.

Таблиця місця розташування (міста, села) Area надає характеристики відповідної сутності за полями:

- id – ідентифікатор місця розташування;
- name – назва місця розташування;

– region_id – ідентифікатор регіону (області), в якому розташовано це місце.

Зв'язком, який з'єднує таблиці Region і Area виступає зв'язок один до багатьох: місце розташування може знаходитись тільки в одній області, область у свою чергу може мати різну кількість таких міст і селищ.

Таблиця регіону Region надає характеристики відповідної сутності за полями:

- id – ідентифікатор регіону (області країни);
- name – назва регіону.

Таблиця поставки (виробництва) Production надає характеристики відповідної сутності за полями:

- id – ідентифікатор поставки;
- advertisement_id – ідентифікатор оголошення, за яким здійснюється ця поставка;
- prod_date – дата поставки;
- quantity – кількість сезонного продукту в поставці;
- unit – одиниця сезонного продукту;
- unit_price – вартість одиниці;
- left_quantity – кількість сезонного продукту, яка ще не була замовлена на даний момент;
- is_preordered – чи можна виконати передзамовлення цієї поставки сезонних продуктів;
- state – поставка завершена чи ще активна.

Зв'язком, який з'єднує таблиці Advertisement і Production виступає зв'язок один до багатьох: поставка може реалізовуватись тільки за одним оголошенням, оголошення у свою чергу може мати різну кількість поставок в різні дати.

Таблиця передзамовлення Preorder надає характеристики відповідної сутності за полями:

- id – ідентифікатор передзамовлення;

- created_at – дата створення передзамовлення;
- needed_at – дата, коли потрібен цей сезонний продукт;
- department_id – ідентифікатор відділення, на яке потрібно доставити цей сезонний продукт;
- quantity – потрібна кількість сезонного продукту;
- price – загальна вартість;
- buyer_id – ідентифікатор покупця;
- production_id – ідентифікатор поставки, до якої належить передзамовлення;
- state – стан передзамовлення;
- sent_at – дата відправлення передзамовлення;
- done_at – дата отримання передзамовлення;
- claim – пояснення проблеми, що виникла під час обслуговування передзамовлення.

Зв'язком, який з'єднує таблиці Department і Preorder виступає зв'язок один до багатьох: передзамовлення може бути відправлено тільки на одне відділення, відділення у свою чергу може отримувати різні передзамовлення за весь час.

Зв'язком, який з'єднує таблиці Buyer і Preorder виступає зв'язок один до багатьох: передзамовлення може бути відправлено тільки одному покупцю, покупець у свою чергу може отримувати різні передзамовлення за весь час.

Зв'язком, який з'єднує таблиці Production і Preorder виступає зв'язок один до багатьох: передзамовлення може стосуватись тільки однієї поставки сезонних продуктів, сама поставка у свою чергу може бути відправлена за різними окремими передзамовленнями.

Таблиця сервісу доставки Delivery надає характеристики відповідної сутності за полями:

- id – ідентифікатор сервісу доставки;
- name – назва сервісу доставки;

– mainsite – вебсайт сервісу доставки.

Таблиця відділення сервісу доставки Department надає характеристики відповідної сутності за полями:

- id – ідентифікатор відділення;
- delivery_id – ідентифікатор сервісу доставки;
- area_id – ідентифікатор місця розташування відділення;
- number – номер відділення сервісу доставки;
- placetype – тип відділення;
- features – опис особливостей відділення.

Зв'язком, який з'єднує таблиці Delivery і Department виступає зв'язок один до багатьох: відділення належить тільки одному сервісу доставки, сам сервіс доставки у свою чергу може мати різні відділення.

Зв'язком, який з'єднує таблиці Area і Department виступає зв'язок один до багатьох: відділення знаходиться тільки в одному місці розташування, конкретному місті або селищі, саме місто або селище може мати різні відділення різних сервісів доставки.

Таблиця оцінювання оголошення Advertisementmark надає характеристики відповідної сутності за полями:

- id – ідентифікатор оцінювання оголошення;
- advertisement_id – ідентифікатор оголошення;
- buyer_id – ідентифікатор покупця;
- marka – оцінка;
- created_at – дата оцінювання;
- descript – текстові особливості.

Зв'язком, який з'єднує таблиці Advertisement і Advertisementmark виступає зв'язок один до багатьох: оцінювання може стосуватися тільки одного оголошення, саме оголошення у свою чергу може мати різні оцінювання.

Зв'язком, який з'єднує таблиці Buyer і Advertisementmark виступає зв'язок один до багатьох: оцінювання може виконуватися тільки одним

покупцем, сам покупець у свою чергу може виконувати різні оцінювання різних оголошень.

Таблиця оцінювання покупця Buyermark надає характеристики відповідної сутності за полями:

- id – ідентифікатор оцінювання покупця;
- farmer_id – ідентифікатор фермера;
- buyer_id – ідентифікатор покупця;
- mark – оцінка;
- created_at – дата оцінювання;
- descript – текстові особливості.

Зв'язком, який з'єднує таблиці Farmer і Buyermark виступає зв'язок один до багатьох: оцінювання може виконуватись тільки одним фермером, сам фермер у свою чергу може мати різні оцінювання.

Зв'язком, який з'єднує таблиці Buyer і Buyermark виступає зв'язок один до багатьох: оцінювання може стосуватися тільки одного покупця, сам покупець у свою чергу може мати різні оцінювання від різних фермерів.

Таблиця листа між фермером і покупцем Letter надає характеристики відповідної сутності за полями:

- id – ідентифікатор листа;
- buyer_id – ідентифікатор покупця;
- farmer_id – ідентифікатор фермера;
- direction – напрям листа, що може бути від покупця до фермера або від фермера до покупця;
- created_at – дата відправлення;
- descript – текст листа.

Зв'язком, який з'єднує таблиці Farmer і Letter виступає зв'язок один до багатьох: лист може бути написаний або адресований тільки одному фермеру, сам фермер у свою чергу може мати різні листи.

Зв'язком, який з'єднує таблиці Buyer і Letter виступає зв'язок один до багатьох: лист може бути написаний або адресований тільки одному покупцю, сам покупець у свою чергу може мати різні листи.

Створена схема БД показана на рис. 2.2.

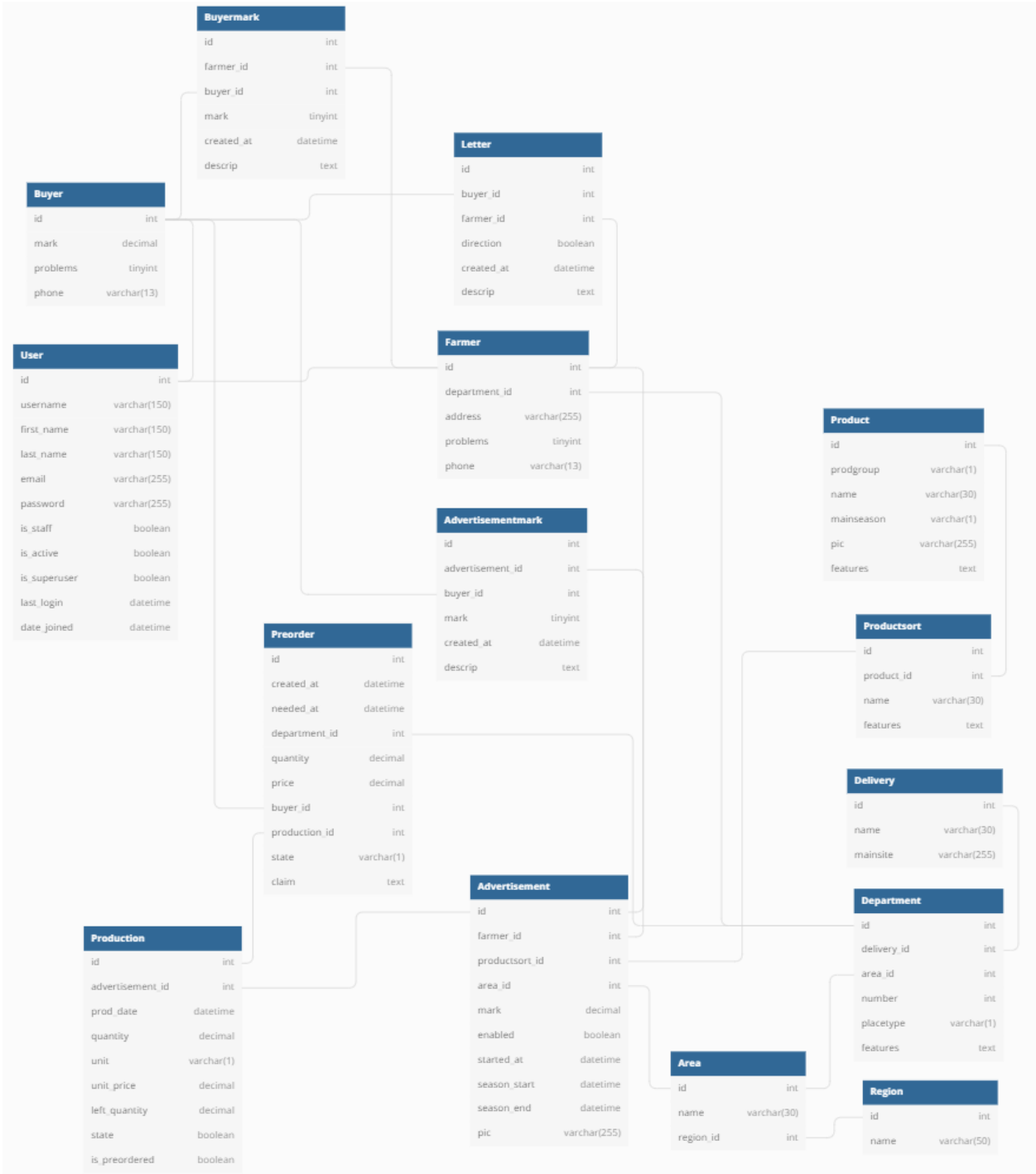


Рисунок 2.2 – Схема БД

2.4 Висновки за розділом 2

Загальну проблему продажу сезонних продуктів, яку було проаналізовано в першому розділі, в другому розділі було доведено до набору функціональних вимог, які було перетворено на сценарії роботи покупців і фермерів. Далі сценарії було використано при побудові схеми бази даних на основі СКБД MySQL, а для реалізації всієї вебсистеми обрано мову Python.

3 РОЗРОБКА ПРОГРАМИ

3.1 Алгоритм функціонування програми

На даному етапі було визначено порядок роботи з функціями програми, які було визначено в розділі 2.

Базовою функцією (рис. 3.1), яка доступна всім користувачам, є пошук сезонних продуктів. Саме тільки цим може займатися користувач до авторизації, тобто до того моменту, як він стане фермером або покупцем. Єдина альтернатива пошуку, яка доступна такому користувачу, це авторизація.

Як тільки авторизація виконана, то потоки подальшої роботи між собою відрізняються в залежності від того, чи авторизувався фермер, чи покупець.

Фермеру доступні можливості, що стосуються керування власними оголошеннями, керування власними поставками, керування передзамовленнями, пошуку сезонних продуктів, листування з покупцями, оцінювання покупців і перегляду оцінювань своїх оголошень. Після виконання кожної такої дії фермер повертається до повторного вибору однієї з цих дій. Фермер також може вийти з акаунту, повернувшись до роботи без авторизації з виконанням пошуку сезонних продуктів.

Покупцю доступні можливості, що стосуються роботи з власними передзамовленнями, пошуку сезонних продуктів, листування з фермерами, оцінювання оголошень продавців і перегляду оцінювань власного профілю, а також керування власним профілем. Після виконання кожної такої дії покупець повертається до повторного вибору однієї з цих дій. Покупець також може вийти з акаунту, повернувшись до роботи без авторизації з виконанням пошуку сезонних продуктів.

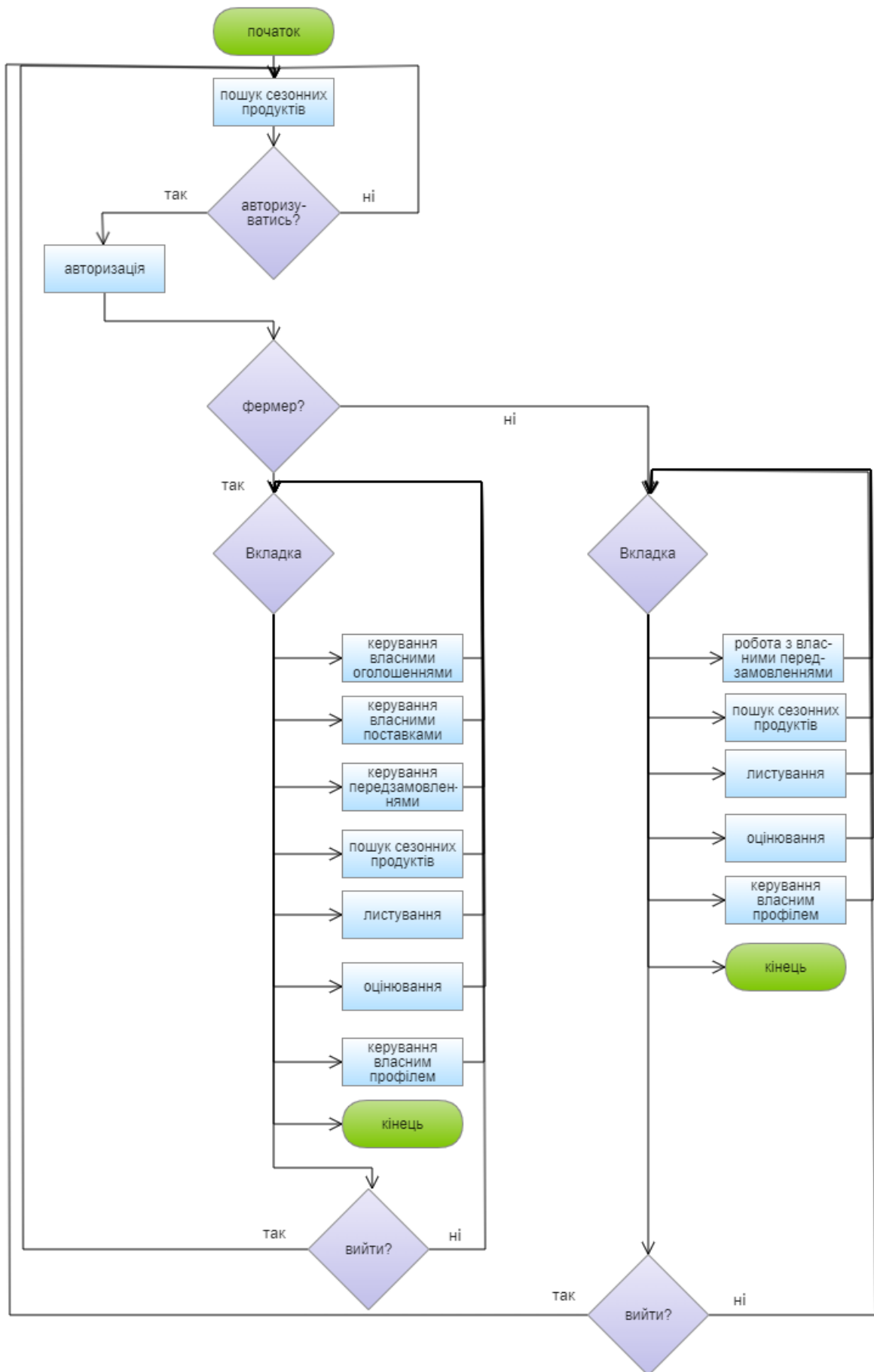


Рисунок 3.1 – Схема функціонування програми

3.2 Опис реалізованих класів програми

При проєктуванні і реалізації класів програми було використано серверний вебфреймворк Django [8]-[9].

Модель оголошення про продаж сезонного продукту реалізована класом Advertisement, що містить поля:

- farmer_id – поле фермера, який постачає сезонні продукти за оголошенням, побудоване на основі зовнішнього ключа з моделі фермера Farmer ForeignKey(Farmer, related_name='farmer_advertisements', on_delete=models.CASCADE), при цьому визначено звернення від фермера до списку його оголошень через додаткову зворотню властивість моделі фермера farmer_advertisements;

- productsort_id – поле сорту сезонного продукту, побудоване на основі зовнішнього ключа з моделі сорту ForeignKey(Productsort, related_name='productsort_advertisements', on_delete=models.CASCADE), при цьому визначено звернення від сорту продукту до списку його оголошень через додаткову зворотню властивість моделі сорту сезонного продукту productsort_advertisements;

- area_id – поле місця розташування, побудоване на основі зовнішнього ключа з моделі місця розташування ForeignKey(Area, related_name='area_advertisements', on_delete=models.CASCADE), при цьому визначено звернення від місця розташування до списку його оголошень через додаткову зворотню властивість моделі місця розташування productsort_advertisements;

- mark – середня оцінка, виставлена покупцями оголошенню, що є полем десяткового числа з фіксованою точністю в 1 цифру після коми і 3 цифри загалом максимально DecimalField(max_digits = 3, decimal_places = 1, default = 0);

- enabled – поле, яке визначає доступний продаж за оголошенням в поточний момент або ні, і є полем логічного типу BooleanField();

- `started_at` – поле дати створення оголошення `DateTimeField()`;
- `season_start` – поле дати початку сезону `DateTimeField()`;
- `season_end` – поле дати завершення сезону `DateTimeField()`;
- `pic` – поле шляху до файлу з фотографією сезонних продуктів за цим оголошенням `FileField(upload_to='images/advertisements/')`.

Окрім полів у складі цього класу було реалізовано декілька методів:

- `__str__(self)` – метод, який дозволяє обробляти об'єкт класу як рядок, повертаючи при цьому назву сезонного продукту та його сорт у вигляді рядка;
- `get_absolute_url(self)` – метод, який повертає шлях, який використовується представленням перегляду оголошення з номером оголошення;
- `save(self, *args, **kwargs)` – метод, який органіює підтримку збереження об'єкту класу, при цьому автоматично заповнюючи дату створення оголошення як поточний момент.

Аналогічно побудовані інші класи моделей з врахуванням структури таблиць, яку було описано в розділі 2. Нижче буде описано тільки певні особливості деяких класів.

Клас `Production` в методі `save` при створенні поставки визначає кількість продукту, що залишилась, `preordered_quantity` у обсязі всієї наявної кількості `quantity`.

Клас `Preorder` в методі `save` визначає дії, необхідні під час збереження передзамовлення. Для цього спочатку визначаються всі передзамовлення, які стосуються цієї поставки, і виділяються з них передзамовлення, що знаходяться у всіх станах окрім створеного, а тоді визначається сумарна кількість продуктів з цих передзамовлень, що дозволяє обчислити кількість продуктів у цій поставці:

```
preorders = Preorder.objects.filter(production_id__id = prod.id)
```

```

preorders = preorders.filter(Q(state = 'C') | Q(state = 'S') | Q(state = 'R') |
Q(state = 'N') | Q(state = 'E') | Q(state = 'P'))
quant = 0
for pr in preorders:
    quant = quant + pr.quantity
prod.left_quantity = prod.quantity - quant

```

Далі обчислюється вартість передзамовлення на основі кількості продуктів у цьому передзамовленні та вартості одиниці даного продукту з цієї поставки:

```
prod.price = prod.quantity * prod.production_id.unit_price
```

Окрім того в методі виконується перевірка всіх передзамовлень зі скаргами для переобчислення кількості проблемних замовлень у фермера та у покупця відповідно.

У класі Advertisementmark метод save виконує переобчислення усередненої оцінки оголошення, до якого стосується остання виставлена оцінка оголошенню, збереження якої відбувається:

```

advert = Advertisement.objects.get(id = self.advertisement_id.id)
marks    =    Advertisementmark.objects.filter(advertisement_id__id    =
advert.id)
avmark = 0
markquant = marks.count()
for m in marks:
    avmark = avmark + m.mark / markquant
advert.mark = avmark

```

Аналогічно в класі Buyermark метод save виконує переобчислення усередненої оцінки покупця, до якого стосується остання виставлена оцінка покупцю від фермера, збереження якої відбувається.

Для визначення обмежених значень певних полів створено відповідні списки кортежів.

Список UNITS визначає набір одиниць вимірювання сезонних продуктів, до яких належать:

- кілограми (значення K відображається як кг);
- грами (значення G відображається як г);
- літри (значення L відображається як л).

Список PRODGROUPS визначає набір груп сезонних продуктів, до яких належать:

- овочі (значення V);
- фрукти (значення F);
- трави (значення G);
- напої (значення D);
- інші продукти (значення O).

Список SEASONS визначає набір сезонів, до яких належать:

- літо (значення S);
- осінь (значення A);
- зима (значення W);
- весна (значення P).

Список PLACETYPES визначає набір типів місць розташування, до яких належать:

- відділення (значення D);
- вантажне відділення (значення L);
- поштомат (значення P).

Список STATES визначає набір станів, до яких належать:

- створено (значення B);
- підтверджено (значення C);

- відправлено (значення S);
- отримано (значення R);
- не отримано (значення N);
- скарга покупця (значення E);
- скарга фермера (значення P).

Для кожної функції програми було розплановано вебсторінку для відображення вмісту або взаємодії з користувачем відповідним чином. Для заповнення даними кожної такої сторінки було розроблено відповідні класи представлень. Шаблони побудовані за допомогою безкоштовного шаблону [10].

Клас `AdvertisementView` організовує підтримку ознайомлення з оголошенням про продаж сезонних продуктів певного сорту на основі створення представлення для заповнення даними шаблону вебсторінки `advert.html`. Окрім даних самого оголошення вносяться дані всіх постачань за цим оголошенням, дані всіх оцінювань цього оголошення та окремо кількість таких оцінювань.

Клас `FarmerAdvertisementsView` організовує підтримку відтворення списку оголошень заданого користувачем фермера. Сортування оголошень відбувається за датою створення, починаючи від поточного моменту.

Клас `NewAdvertisementView` організовує підтримку створення нового оголошення на основі стандартного класу суміші попередньої авторизації `LoginRequiredMixin` і класу представлення створення `CreateView`. Для представлення через параметри передається сорт сезонного продукту та місце розташування, де виробляється продукт. При збереженні програмно вноситься інформація про фермера на основі даних авторизованого користувача.

Клас `UpdateAdvertisementView` організовує підтримку оновлення існуючого оголошення на основі стандартного класу суміші попередньої авторизації `LoginRequiredMixin` і класу представлення оновлення

UpdateView. Для представлення через параметри передається сорт сезонного продукту та місце розташування, де виробляється продукт.

Клас MyAdvertisementsView організовує підтримку відтворення списку власних оголошень авторизованого користувача фермера. Сорткування оголошень відбувається за датою створення, починаючи від поточного моменту.

Клас OwnAdvertisementView організовує підтримку ознайомлення з власним оголошенням про продаж сезонних продуктів певного сорту на основі створення представлення для заповнення даними шаблону вебсторінки ownadvert.html. Окрім даних самого оголошення вносяться дані всіх постачань за цим оголошенням, дані всіх оцінювань цього оголошення, окремо кількість таких оцінювань та кількість передзамовлень, оформлених за цим оголошенням.

Клас NewProductionView організовує підтримку створення нового постачання за оголошенням на основі стандартного класу суміші попередньої авторизації LoginRequiredMixin і класу представлення створення CreateView. Для представлення через параметри передається оголошення, до якого належить постачання.

Клас UpdateProductionView організовує підтримку оновлення існуючого постачання за оголошенням на основі стандартного класу суміші попередньої авторизації LoginRequiredMixin і класу представлення оновлення UpdateView. Для представлення через параметри передається оголошення, до якого належить постачання.

Клас ProductionView організовує підтримку ознайомлення з постачанням сезонних продуктів певного сорту на основі створення представлення для заповнення даними шаблону вебсторінки product.html. Окрім даних самого постачання вносяться дані всіх передзамовлень за цим оголошенням, кількість таких передзамовлень.

Клас MyProductionsView організовує підтримку відтворення списку власних постачань авторизованого користувача фермера. Сорткування

постачань відбувається за датою постачання, починаючи від поточного моменту.

Клас `AdvertisementProductionsView` організовує підтримку відтворення списку постачань за оголошенням. Сорткування постачань відбувається за датою постачання, починаючи від поточного моменту. В якості параметра через GET-запит передається оголошення, постачання до якого потрібно відтворити.

Клас `NewPreorderView` організовує підтримку створення нового передзамовлення на основі стандартного класу суміші попередньої авторизації `LoginRequiredMixin` і класу представлення створення `CreateView`. Для представлення через параметри передається постачання, якому відповідає передзамовлення, та відділення сервісу доставки, куди потрібно надіслати передзамовлення.

Клас `UpdatePreorderView` організовує підтримку оновлення передзамовлення на основі стандартного класу суміші попередньої авторизації `LoginRequiredMixin` і класу представлення оновлення `UpdateView`. Для представлення через параметри передається постачання та відділення служби доставки. При збереженні програмно вноситься інформація про покупця на основі даних авторизованого користувача.

Клас `FarmerPreordersView` організовує підтримку відтворення списку передзамовлень, що стосуються авторизованого фермера. Сорткування постачань відбувається за датою створення передзамовлення, починаючи від поточного моменту. В якості параметра через GET-запит передається стан передзамовлення, який необхідно відтворити.

Клас `MyPreordersView` організовує підтримку відтворення списку передзамовлень авторизованого покупця. Сорткування постачань відбувається за датою створення передзамовлення, починаючи від поточного моменту. В якості параметра через GET-запит передається стан передзамовлення, який необхідно відтворити.

Клас `NewAdvertisementmarkView` організовує підтримку створення нового оцінювання оголошення на основі стандартного класу суміші попередньої авторизації `LoginRequiredMixin` і класу представлення створення `CreateView`. Для представлення через параметри передається оголошення, за яким відбувається оцінювання. При збереженні програмно вноситься інформація про покупця, який виконав оцінювання, на основі даних авторизованого користувача.

Клас `UpdateAdvertisementmarkView` організовує підтримку оновлення оцінювання оголошення на основі стандартного класу суміші попередньої авторизації `LoginRequiredMixin` і класу представлення оновлення `UpdateView`. Для представлення через параметри передається оголошення.

Клас `ProductionSearchView` організовує підтримку відтворення списку результатів пошуку поставачань. Параметри пошуку поставачань передаються через GET-запит: період дат, коли потрібні продукти, область, де покупець хоче забрати продукти самостійно, а також сорт продукту.

Клас `AdvertisementSearchView` організовує підтримку відтворення списку результатів пошуку оголошень. Параметри пошуку оголошень передаються через GET-запит:

- початок періоду, коли потрібні продукти;
- завершення періоду, коли потрібні продукти;
- область, де покупець хоче забрати продукти самостійно;
- місце розташування, де покупець готовий самостійно забрати продукти;
- назва продукту, за якою перевіряється сам продукт та сорт продукту.

Усі параметри окрім назви продукту можуть бути пропущені.

Клас `FarmerPopularAreasView` організовує підтримку відтворення списку популярних місць, за якими відправляє передзамовлення фермер. За кожним місцем вказується кількість передзамовлень, які було оформлені за даним місцем.

Клас `MyAdvertisementmarksView` організовує підтримку відтворення списку оцінок оголошень, створених авторизованим покупцем. Сорткування оцінювань відбувається за датою, починаючи від поточного моменту.

Клас `MyMarksView` організовує підтримку відтворення списку оцінок оголошень, які належать авторизованому фермеру. Сорткування оцінювань відбувається за датою, починаючи від поточного моменту.

Клас `FarmerPopularProdsortsView` організовує підтримку відтворення списку кількості проданих продуктів за кожним сортом сезонних продуктів, які продає авторизований фермер, сортуючи їх за кількістю.

Клас `NewBuyermarkView` організовує підтримку створення нового оцінювання покупця авторизованим фермером на основі стандартного класу суміші попередньої авторизації `LoginRequiredMixin` і класу представлення створення `CreateView`. Для представлення через параметри передається покупець, оцінювання якого відбувається. При збереженні програмно вноситься інформація про фермера, який виконав оцінювання, на основі даних авторизованого користувача.

Клас `UpdateBuyermarkView` організовує підтримку оновлення оцінювання покупця на основі стандартного класу суміші попередньої авторизації `LoginRequiredMixin` і класу представлення оновлення `UpdateView`. Для представлення через параметри передається покупець.

Клас `MyBuyermarksView` організовує підтримку відтворення списку оцінок покупців, створених авторизованим фермером. Сорткування оцінювань відбувається за датою, починаючи від поточного моменту.

Клас `MyBuyMarksView` організовує підтримку відтворення списку оцінок щодо авторизованого покупця. Сорткування оцінювань відбувається за датою, починаючи від поточного моменту.

Клас `NewLetterView` організовує підтримку створення нового листа на основі стандартного класу суміші попередньої авторизації `LoginRequiredMixin` і класу представлення створення `CreateView`. Для представлення через параметри передається адресат листа, напрямок (від

покупця фермеру або від фермера покупцю). При збереженні програмно вноситься інформація про автора листа на основі даних авторизованого користувача. При цьому визначається, чи автор є фермером, чи покупцем, в залежності від цього заповнюється автоматично одне поле, а інше – через заданий параметр.

Клас `BuyerView` організовує підтримку ознайомлення з профілем покупця на основі створення представлення для заповнення даними шаблону вебсторінки `bprofile.html`. Окрім даних самого покупця вносяться дані оцінок, виставлених покупцю, та кількість таких оцінок.

Клас `FarmerView` організовує підтримку ознайомлення з профілем фермера на основі створення представлення для заповнення даними шаблону вебсторінки `mprofile.html`.

Клас `LettersView` організовує підтримку відтворення списку повідомлень між авторизованим користувачем і користувачем, переданим через параметр GET-запиту.

Клас `AreaFarmersView` організовує підтримку відтворення списку фермерів, які діють в місці розташування, переданому через параметр GET-запиту.

3.3 Висновки за розділом 3

За схемою БД було розроблено моделі вебсистеми, визначено послідовність роботи програми над виконанням описаних функцій, структуровано вебсистему з визначенням вебсторінок з відповідними шаблонами і представленнями, і реалізовано програму в цілому. Розроблені програмні одиниці було описано разом з логікою їх роботи.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Призначенням програми є створення майданчика для взаємодії фермерів, садівників, аграріїв, які є представниками малого бізнесу або просто мають свою власну невелику ділянку, з покупцями, які бажають знайти якісні сезонні продукти в своєму регіоні або отримати їх за допомогою сервісу доставки.

4.2 Умови виконання програми

Мінімальний рівень технічних засобів для забезпечення роботи програмної системи включає процесор не менш ніж 4 ядра і частотою 2,9 ГГц, оперативну пам'ять у 8 Гб, жорсткий диск у 1 Тб.

4.3 Експлуатація програми

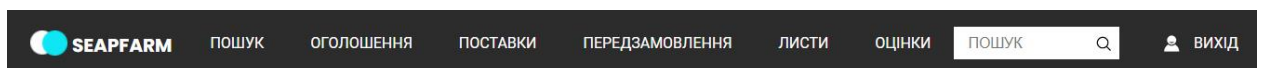
Експлуатація програми відбувається у відповідності до схеми функціонування програми, що була представлена в розділі 3.

Розглянемо засоби роботи фермера. Його меню включає такі пункти:

- пошук – дозволяє виконати широкий пошук сезонних продуктів, якщо це необхідно фермеру, на відміну від рядка пошуку в меню, яке дозволяє виконати пошук тільки за назвою продукту або назвою сорту продукту;

- оголошення – дозволяє переглянути список оголошень фермера (рис. 4.1), звідки можна перейти до редагування обраного оголошення: кожне оголошення відображається зображенням сорту продукту, його назвою та округленою оцінкою, надаються також кнопки для відкриття оголошення та переходу до списку поставок за оголошенням;

- поставки – дозволяє керувати всіма власними поставками за власними оголошеннями одразу, в тому числі створювати нові;
- передзамовлення – дозволяє керувати всіма оформленими покупцями передзамовленнями, а також аналізувати узагальнені дані за ними та створювати оцінки за передзамовленнями покупцям;
- листи – дозволяє переглягнути листи від покупців та відповісти на них;
- оцінки – дозволяє переглянути оцінки власним оголошенням (вони також доступні зі сторінки кожного оголошення);
- профіль – відображається на місці іконки користувача, відкривається сторінка редагування даних профілю;
- вихід.



ОГОЛОШЕННЯ ФЕРМИ

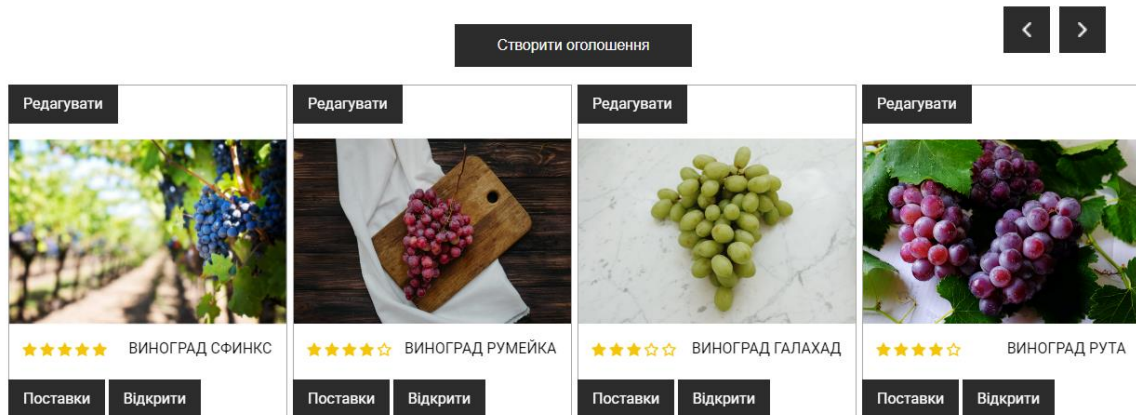


Рисунок 4.1 – Відтворення списку власних оголошень

Меню покупця має однакові пункти з фермером, а загалом включає в себе наступні:

- пошук – дозволяє виконати пошук сезонних продуктів (рис. 4.2);

- передзамовлення – дозволяє переглянути власні передзамовлення, їх стан, створити оцінки за передзамовленнями;
- листи – дозволяє переглянути листи, отримані у відповідь від фермерів, а також створити листи до них;
- оцінки – дозволяє переглянути власні виставлені оцінки оголошенням, а також оцінки, виставлені від фермерів;
- профіль – відображається на місці іконки користувача, відкривається сторінка редагування даних профілю;
- вихід.

The screenshot shows the SEAPFARM website's search interface. At the top is a dark navigation bar with the SEAPFARM logo and links for ПОШУК, ПЕРЕДЗАМОВЛЕННЯ, ЛИСТИ, ОЦІНКИ, and a search bar with the word ПОШУК. Below this is a section titled ШУКАТИ СЕЗОННІ ПРОДУКТИ. The search form is divided into two columns. The left column contains: 'Тип' (Oголошення), 'Продукт' (empty text field), 'Кількість продукту' (empty text field), 'Найраніша дата' (16.06.2022 with a calendar icon), and 'Найпізніша дата' (16.06.2022 with a calendar icon). The right column contains: 'Область' (Запорізька), 'Місце розташування' (Будь-яке), 'Мінімальна оцінка' (0), and 'Максимальна оцінка' (5). All dropdown menus have a downward arrow icon. At the bottom left of the form is a black button with the text 'Шукати'.

Рисунок 4.2 – Пошук сезонних продуктів

Під час пошуку сезонних продуктів користувач (це стосується і покупця, і неавторизованого користувача, і фермера) може обрати тип пошуку (пошук постачань конкретних або пошук загалом оголошень – другий варіант дозволяє шукати зокрема оголошення, а потім вже, знаючи джерело продуктів, у потрібний момент звертатися до цього фермера), продукт (пошук за цим полем відбувається і в назві продукту, і в сорті продукту), потрібну кількість продукту (задається в кілограмах або літрах, залежно від продукту), найранішу дату, коли потрібно закупити продукт, та

найпізнішу дату, область, де розташовано ферму, місце розташування, мінімальні та максимальну оцінку оголошення (з кроком 0,5).

У результатах пошуку (рис. 4.3) зазначається список оголошень, де за кожним виводиться фотографія, оцінка, вартість за одиницю з відповідної поставки. За кожним оголошенням можна відкрити його, перейшовши на повну сторінку оголошення (там зокрема зазначається конкретна ферма, оцінки оголошення), або створити одразу замовлення за допомогою кнопки «Замовити».

РЕЗУЛЬТАТИ ПОШУКУ

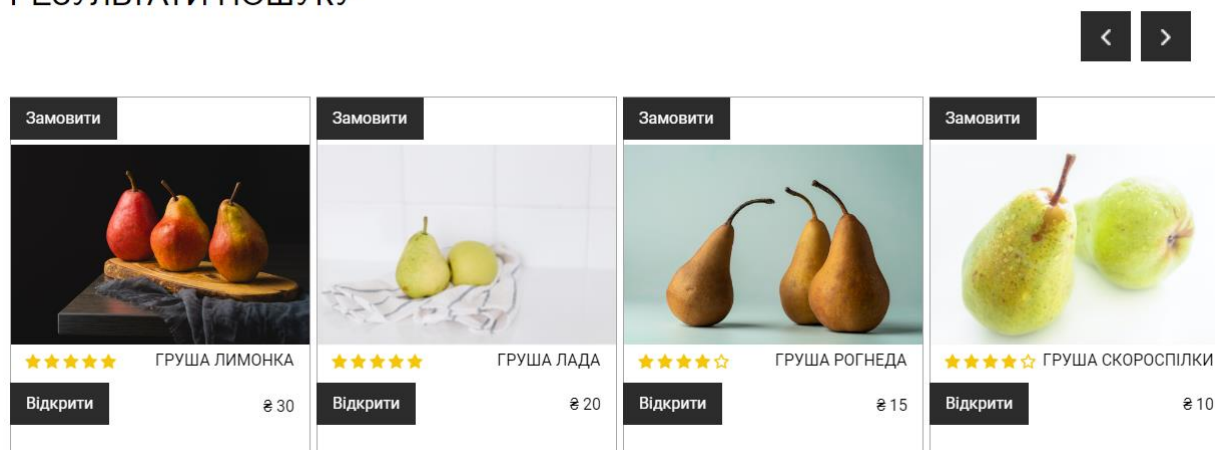


Рисунок 4.3 – Результати пошуку сезонних продуктів

Після вибору таким чином продукту і ферми, з якої цей сорт продукту походить, відкривається сторінка створення нового передзамовлення (рис. 4.4).

На вебсторінці створення нового замовлення можна задати бажану дату отримання посилки з продуктами, потрібну кількість продукту, обрати службу доставки, місце доставки та конкретне відділення. Якщо зі списку служби доставки обрати значення «Самостійно», то передбачається, що покупець приїде на ферму і забере самостійно продукти. З цієї сторінки можна також повернутися до сторінки оголошення, натискаючи на назву продукту і сорту.



ПОШУК

ПЕРЕДЗАМОВЛЕННЯ

ЛИСТИ

ОЦІНКИ

ПОШУК

ВИХІД

СТВОРИТИ ЗАМОВЛЕННЯ

Бажана дата отримання

16.06.2022

Кількість продукту, кг

3

Служба доставки

Нова пошта

Місце доставки

Запоріжжя

Відділення

Відділення № 11

Вартість: 90 грн.

Передзамовити

Груша Лимонка

Лимонарій (Запоріжжя)



Рисунок 4.4 – Створення замовлення

4.4 Тестування програми

Тестування програми виконувалось з перевіркою функцій програми, які вона повинна була виконувати, з додатковою перевіркою кросбраузерності. Перевірка відбувалась на мобільних пристроях під керуванням операційної системи Android, на ноутбуку під керуванням операційної системи Windows 10 через браузері Mozilla Firefox та Google Chrome останніх версій на поточний момент.

4.5 Висновки за розділом 4

Описано процес роботи з програмою з деталізацією основних режимів роботи. Програму протестовано з успішним завершенням.

ВИСНОВКИ

Мету розробити програмне забезпечення пошуку сезонних продуктів для підтримки прямої взаємодії між фермерами і покупцями було досягнуто в роботі.

Загальну проблему продажу сезонних продуктів спочатку було проаналізовано, на основі чого сформовано функціональні вимоги, які було перетворено на сценарії роботи покупців і фермерів. Далі сценарії було використано при побудові схеми бази даних, за якою було розроблено моделі вебсистеми, визначено послідовність роботи програми над виконанням описаних функцій, структуровано вебсистему з визначенням вебсторінок з відповідними шаблонами і представленнями, і реалізовано програму в цілому. Розроблені програмні одиниці було описано разом з логікою їх роботи.

Основними засобами розробки були мова Python, серверний вебфреймворк Django, система керування базами даних MySQL. Програму також було перевірено шляхом тестування.

Розроблена програма дозволяє створити майданчик для прямої взаємодії між фермерами і покупцями, які знаходяться в пошуку сезонних продуктів, для уникнення третіх сторін і отримання якісних свіжих продуктів. При реалізації використано ідею, що продукти можуть пересилатися сервісами доставки, а можуть надаватися прямо на фермі покупцю, якщо він готовий самостійно дістатися її. Окрім того за необхідності фермери можуть самостійно розподіляти наявну кількість продуктів між покупцями за допомогою програми. Для захисту інтересів обох сторін призначене оцінювання, яке дозволяє встановлювати проблеми під час продажів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Goolsby, J. M. 4 Reasons To Eat By The Seasons. What Is In Season Right Now [Electronic resource] / J. M. Goolsby, M. Fahey. – Access mode : <https://www.mindbodygreen.com/0-4807/10-Reasons-To-Eat-Whats-In-Season.html>.
2. 5 Reasons to Eat Local, Seasonal Food. Sto Food Wate [Electronic resource]. – Access mode : <https://stopfoodwaste.ie/resource/5-reasons-to-eat-local-seasonal-food>.
3. Seasonal Food Guide [Electronic resource]. – Access mode : <https://www.seasonalfoodguide.org/why-eat-seasonally>.
4. The Sharing Farm Society. Volunteers growing and harvesting fruits and vegetables – for our neighbours in need [Electronic resource]. – Access mode : <https://www.sharingfarm.ca/>.
5. Shaw, Z. Learn Python 3 the Hard Way [Text] : A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code / Z. Shaw. – Boston, Massachusetts : Addison-Wesley Professional, 2017. – 320 p.
6. Ramalho, L. Fluent Python [Text] : Clear, Concise, and Effective Programming / L. Ramalho. – Massachusetts : O'Reilly Media, 2022. – 1012 p.
7. Evans, B. Java in a Nutshell [Text] : A Desktop Quick Reference / B. Evans, D. Flanagan. – Massachusetts : O'Reilly Media, 2019. – 455 p.
8. Percival, H. Test-Driven Development with Python [Text] : Obey the Testing Goat: Using Django, Selenium, and JavaScript / H. Percival. – Massachusetts : O'Reilly Media, 2017. – 624 p.
9. Mele, A. Django 3 By Example [Text] : Build powerful and reliable Python web applications from scratch, 3rd Edition / A. Mele. – Birmingham : Packt Publishing, 2020. – 568 p.
10. Medion Free Website Template. Free CSS Templates. Free CSS [Electronic resource]. – Access mode : <https://www.free-css.com/free-css-templates/page279/medion>.

ДОДАТОК А
Технічне завдання

Вступ

Свіжі сезонні продукти є основою здорового харчування. На пошук таких продуктів і замовлення їх безпосередньо у виробника (фермера, садівника) і направлена програма. Користувач має знайти потрібні йому продукти, а тоді поїхати їх забрати власноруч або отримати за допомогою сервісу доставки. Такий підхід дозволить розвиватися динамічно малому бізнесу, а покупцям отримувати вчасно якісний продукт.

A.1 Підстави для розробки

Розробка в межах дипломної кваліфікаційної роботи реалізується на підставі наказу № 102 від 27 квітня 2022 р. за Національним університетом «Запорізька політехніка» за темою «Програмне забезпечення пошуку сезонних продуктів».

A.2 Призначення розробки

Призначенням розробки є створення майданчика для взаємодії фермерів, садівників, аграріїв, які є представниками малого бізнесу або просто мають свою власну невелику ділянку, з покупцями, які бажають знайти якісні сезонні продукти в своєму регіоні або отримати їх за допомогою сервісу доставки.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Функціональні характеристики, які повинна мати програма, охоплюють:

- пошук оголошень про продаж потрібних сезонних продуктів;

- відтворення власних оголошень;
- визначення оголошення;
- оновлення оголошення;
- передзамовлення сезонних продуктів;
- керування передзамовленням сезонних продуктів;
- оцінювання покупця;
- оцінювання сезонних продуктів за оголошенням;
- ознайомлення з оголошенням про продаж сезонного продукту;
- відтворення оголошень фермера;
- виставлення нової поставки продуктів за оголошенням;
- оновлення поставки;
- ознайомлення з поставкою продуктів за оголошенням;
- відтворення власних поставок;
- відтворення поставок за оголошенням;
- оновлення передзамовлення;
- відтворення передзамовлень, адресованих фермеру;
- відтворення власних передзамовлень;
- пошук потрібних сезонних продуктів;
- аналіз кількості передзамовлень у фермера за місцем;
- відтворення списку власних оцінювань оголошень;
- відтворення списку оцінювань оголошень фермера;
- аналіз обсягів замовлень сортів сезонних продуктів у фермера;
- авторизація, реєстрація;
- відтворення списку власних оцінювань покупців;
- відтворення списку оцінювань покупця;
- надсилання листа;
- визначення скарги на передзамовлення;
- відтворення списку листів між користувачами;
- відтворення списку продавців за місцем розташування;
- ознайомлення з профілем покупця;

– ознайомлення з профілем фермера.

А.3.2 Умови експлуатації

Експлуатація програми має забезпечуватися з одного боку звичайними користувачами, які працюють з даними, наповнюють їх, а з іншого боку – адміністраторами системи та БД.

А.3.3 Вимоги до складу та параметрів технічних засобів

Мінімальний рівень технічних засобів для забезпечення роботи програмної системи включає процесор не менш ніж 4 ядра і частотою 2,9 ГГц, оперативну пам'ять у 8 Гб, жорсткий диск у 1 Тб.

А.4 Порядок контролю та приймання

Для контролю виконання роботи використовується календарний план, а приймання роботи виконується під час проведення процедури захисту роботи перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

Б.1 Текст файла models.py

```
from django.contrib.auth.models import AbstractUser
from django.db import models
from django.utils import timezone
from django.urls import reverse
```

```
UNITS = [
    ('K', 'кг'),
    ('G', 'г'),
    ('L', 'л'),
]
```

```
PRODGROUPS = [
    ('V', 'овочі'),
    ('F', 'фрукти'),
    ('G', 'трави'),
    ('D', 'напої'),
    ('O', 'інше'),
]
```

```
SEASONS = [
    ('S', 'літо'),
    ('A', 'осінь'),
    ('W', 'зима'),
    ('P', 'весна'),
]
```

```
PLACETYPES = [
    ('D', 'відділення'),
    ('L', 'вантажне відділення'),
    ('P', 'поштомат'),
]
```

```
STATES = [
    ('B', 'створено'),
    ('C', 'підтверджено'),
    ('S', 'відправлено'),
    ('R', 'отримано'),
    ('N', 'не отримано'),
    ('E', 'скарга покупця'),
    ('P', 'скарга фермера'),
]
```

```

class Area (models.Model):
    name = models.CharField(max_length=30)
    region_id = models.ForeignKey(Region,
related_name='region_areas', on_delete=models.CASCADE)

    def __str__(self):
        return self.name + ' (' + self.region_id.name + ')'

class Region (models.Model):
    name = models.CharField(max_length=50)

    def __str__(self):
        return self.name

class Advertisement (models.Model):
    farmer_id = models.ForeignKey(Farmer,
related_name='farmer_advertisements', on_delete=models.CASCADE)
    productsort_id = models.ForeignKey(Productsort,
related_name='productsort_advertisements', on_delete=models.CASCADE)
    area_id = models.ForeignKey(Area,
related_name='area_advertisements', on_delete=models.CASCADE)
    mark = models.DecimalField(max_digits = 3, decimal_places =
1, default = 0)
    enabled = models.BooleanField()
    started_at = models.DateTimeField()
    season_start = models.DateTimeField()
    season_end = models.DateTimeField()
    pic = models.FileField(upload_to='images/advertasements/')

    def __str__(self):
        return self.productsort_id.product_id.name + ' ' +
self.productsort_id.name

    def get_absolute_url(self):
        return reverse('advertisementview', kwargs={'pk':
self.pk})

    def save(self, *args, **kwargs):
        self.started_at = datetime.now
        super(Advertisement, self).save(*args, **kwargs)

class Production (models.Model):
    advertisement_id = models.ForeignKey(Advertisement,
related_name='advert_productions', on_delete=models.CASCADE)
    prod_date = models.DateTimeField()

```

```

        quantity = models.DecimalField(max_digits = 10,
decimal_places = 2, default = 0)
        unit = models.CharField(max_length=1, choices=UNITS,
default='G')
        unit_price = models.DecimalField(max_digits = 10,
decimal_places = 2, default = 0)
        left_quantity = models.DecimalField(max_digits = 10,
decimal_places = 2, default = 0)
        state = models.BooleanField()
        is_preordered = models.BooleanField()

    def __str__(self):
        return
self.advertisement_id.productsort_id.product_id.name + ' ' + self.
advertisement_id.productsort_id.name

    def get_absolute_url(self):
        return reverse('productionview', kwargs={'pk':
self.pk})

    def save(self, *args, **kwargs):
        self.preordered_quantity = self.quantity
        super(Production, self).save(*args, **kwargs)

class Preorder (models.Model):
    production_id = models.ForeignKey(Production,
related_name='production_preorders', on_delete=models.CASCADE)
    department_id = models.ForeignKey(Department,
related_name='department_preorders', on_delete=models.CASCADE)
    buyer_id = models.ForeignKey(Buyer,
related_name='buyer_preorders', on_delete=models.CASCADE)
    created_at = models.DateTimeField()
    needed_at = models.DateTimeField()
    price = models.DecimalField(max_digits = 10, decimal_places
= 2, default = 0)
    quantity = models.DecimalField(max_digits = 10,
decimal_places = 2, default = 0)
    state = models.CharField(max_length=1, choices=STATES,
default='B')
    claim = models.TextField(blank = True)

    def get_absolute_url(self):
        return reverse('preorderview', kwargs={'pk': self.pk})

```

```

    def save(self, *args, **kwargs):
        self.created_at = datetime.now
        prod = Production.objects.get(id =
self.production_id.id)
        preorders = Preorder.objects.filter(production_id__id
= prod.id)
        preorders = preorders.filter(Q(state = 'C') | Q(state
= 'S') | Q(state = 'R') | Q(state = 'N') | Q(state = 'E') | Q(state =
'P'))
        quant = 0
        for pr in preorders:
            quant = quant + pr.quantity
        prod.left_quantity = prod.quantity - quant
        prod.price = prod.quantity *
prod.production_id.unit_price
        prod.save()
        if self.state == 'E':
            farmer =
self.production_id.advertisement_id.farmer_id
            precount =
Preorder.objects.filter(self.production_id.advertisement_id.farmer_id_
_id = farmer.id)
            precount = precount.filter(state = 'E').count()
            farmer.problems = precount
            farmer.save()
        elif self.state == 'P':
            buy = self.buyer_id.
            precount = Preorder.objects.filter(buyer_id__id =
buy.id)
            precount = precount.filter(state = 'P').count()
            buy.problems = precount
            buy.save()
        super(Preorder, self).save(*args, **kwargs)

class Product (models.Model):
    prodgroup = models.CharField(max_length=1,
choices=PRODGROUPS, default='0')
    name = models.CharField(max_length=30)
    mainseason = models.CharField(max_length=1, choices=SEASONS,
default='S')
    pic = models.FileField(upload_to='images/products/')
    features = models.TextField(blank = True)

    def __str__(self):
        return self.name

```

```

    def get_absolute_url(self):
        return reverse('productview', kwargs={'pk': self.pk})

class Productsort (models.Model):
    product_id = models.ForeignKey(Product,
related_name='product_sorts', on_delete=models.CASCADE)
    name = models.CharField(max_length=30)
    features = models.TextField(blank = True)

    def __str__(self):
        return self.name

    def get_absolute_url(self):
        return reverse('productsortview', kwargs={'pk':
self.pk}))

class Delivery (models.Model):
    name = models.CharField(max_length=30)
    mainsite = models.CharField(max_length=255)

class Department (models.Model):
    delivery_id = models.ForeignKey(Delivery,
related_name='product_sorts', on_delete=models.CASCADE)
    area_id = models.ForeignKey(Area,
related_name='product_sorts', on_delete=models.CASCADE)
    number = models.IntegerField()
    placetype = models.CharField(max_length=1,
choices=PLACETYPES, default='D')
    features = models.TextField(blank = True)

    def __str__(self):
        return self.delivery_id.name + ' №' + self.number

    def get_absolute_url(self):
        return reverse('departmentview', kwargs={'pk':
self.pk}))

class Advertisementmark (models.Model):
    advertisement_id = models.ForeignKey(Advertisement,
related_name='advert_marks', on_delete=models.CASCADE)
    buyer_id = models.ForeignKey(Buyer,
related_name='buyer_marks', on_delete=models.CASCADE)
    mark = models.SmallIntegerField (default = 0)
    created_at = models.DateTimeField()

```

```

        descrip = models.TextField(blank = True)

    def get_absolute_url(self):
        return reverse('advertmarkview', kwargs={'pk':
self.pk}))

    def save(self, *args, **kwargs):
        self.created_at = datetime.now
        advert = Advertisement.objects.get(id =
self.advertisement_id.id)
        marks =
Advertisementmark.objects.filter(advertisement_id__id = advert.id)
        avmark = 0
        markquant = marks.count()
        for m in marks:
            avmark = avmark + m.mark / markquant
        advert.mark = avmark
        advert.save()
        super(Advertisementmark, self).save(*args, **kwargs)

class Buyermark (models.Model):
    farmer_id = models.ForeignKey(Farmer,
related_name='farmer_buyermarks', on_delete=models.CASCADE)
    buyer_id = models.ForeignKey(Buyer,
related_name='buyer_marks', on_delete=models.CASCADE)
    mark = models.SmallIntegerField(default = 0)
    created_at = models.DateTimeField()
    descrip = models.TextField(blank = True)

    def get_absolute_url(self):
        return reverse('buyermarkview', kwargs={'pk':
self.pk}))

    def save(self, *args, **kwargs):
        self.created_at = datetime.now
        buy = Buyer.objects.get(id = self.buyer_id.id)
        marks = Buyermark.objects.filter(buyer_id__id =
buy.id)

        avmark = 0
        markquant = marks.count()
        for m in marks:
            avmark = avmark + m.mark / markquant
        buy.mark = avmark
        buy.save()
        super(Buyermark, self).save(*args, **kwargs)

```

```

class Letter (models.Model):
    farmer_id = models.ForeignKey(Farmer,
related_name='farmer_buyermarks', on_delete=models.CASCADE)
    buyer_id = models.ForeignKey(Buyer,
related_name='buyer_marks', on_delete=models.CASCADE)
    direction = models.BooleanField()
    created_at = models.DateTimeField()
    descrip = models.TextField(blank = True)

    def save(self, *args, **kwargs):
        self.created_at = datetime.now
        super(Letter, self).save(*args, **kwargs)

class Farmer (AbstractUser):
    department_id = models.ForeignKey(Department,
related_name='department_farmers', on_delete=models.CASCADE)
    address = models.CharField(max_length=255)
    problems = models.SmallIntegerField(default = 0)
    phone = models.CharField(max_length=13)

    def __str__(self):
        return self.last_name + ' ' + self.first_name

class Buyer (AbstractUser):
    mark = models.DecimalField(max_digits = 3, decimal_places =
1, default = 0)
    problems = models.SmallIntegerField(default = 0)
    phone = models.CharField(max_length=13)

    def __str__(self):
        return self.last_name + ' ' + self.first_name

```

Б.2 Текст файла advertisementview.py

```

from django.contrib.auth.mixins import LoginRequiredMixin
from django.urls import reverse_lazy
from django.views.generic import ListView
from django.views.generic import UpdateView
from django.views.generic.edit import CreateView
from django.views.generic.detail import DetailView
from datetime import datetime
from .models import Advertisement
from .models import Product
from .models import Area

```

```

from .models import Region
from .models import Productsort
from .models import Farmer
from .models import Production
from .models import Department
from .models import Preorder
from .models import Buyer
from .models import Buyermark
from .models import Letter
from .models import Advertisementmark

class AdvertisementView (DetailView):
    template_name = 'advert.html'
    model = Advertisement

    def get_context_data(self, **kwargs):
        advdata = super().get_context_data(**kwargs)
        adv = get_object_or_404(Advertisement, id =
self.object.pk)
        prods = adv.advert_productions
        prods = prods.filter(state = True)
        advdata ['productions'] = prods
        ams =
Advertisementmark.objects.filter(advertisement_id__id = adv.id)
        advdata ['marks'] = ams
        advdata ['marksquant'] = ams.count()
        return advdata

class FarmerAdvertisementsView(ListView):
    model = Advertisement
    template_name = 'farmeradverts.html'
    paginate_by = 10

    def get_queryset(self):
        farmer = self.request.GET.get('farmer')
        adverts = Advertisement.objects.filter(farmer_id__id =
farmer)

        adverts = adverts.filter(enabled = True)
        adverts = adverts.order_by('-started_at')
        return adverts

    def get_context_data(self, **kwargs):
        advdata = super(FarmerAdvertisementsView,
self).get_context_data(**kwargs)

```

```

        farmer = self.request.GET.get('farmer')
        advdata ['farmer_id'] = farmer
        return advdata

class NewAdvertisementView(LoginRequiredMixin, CreateView):
    model = Advertisement
    fields = ['productsort_id', 'area_id', 'mark', 'enabled',
'season_start', 'season_end', 'pic']
    template_name = 'newadvert.html'

    def get_success_url(self):
        return reverse_lazy('advert-new', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        self.productsort_id = get_object_or_404(Productsort,
id=self.kwargs['productsort_id'])
        kwargs['productsort_id'] = self.productsort_id
        self.area_id = get_object_or_404(Area,
id=self.kwargs['area_id'])
        kwargs['area_id'] = self.area_id
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.productsort_id = get_object_or_404(Productsort,
id=self.kwargs['productsort_id'])
        form.instance.productsort_id = self.productsort_id
        self.area_id = get_object_or_404(Area,
id=self.kwargs['area_id'])
        form.instance.area_id = self.area_id
        farmer = Farmer.objects.get(id = user.id)
        form.instance.farmer_id = farmer
        return super().form_valid(form)

class UpdateAdvertisementView(LoginRequiredMixin, UpdateView):
    model = Advertisement
    fields = ['productsort_id', 'area_id', 'mark', 'enabled',
'season_start', 'season_end', 'pic']
    template_name = 'updateadvert.html'

    def get_success_url(self):
        return reverse_lazy('advert-update', kwargs={'pk':
self.object.pk})

```

```

    def get_context_data(self, **kwargs):
        self.productsort_id = get_object_or_404(Productsort,
id=self.kwargs['productsort_id'])
        kwargs['productsort_id'] = self.productsort_id
        self.area_id = get_object_or_404(Area,
id=self.kwargs['area_id'])
        kwargs['area_id'] = self.area_id
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.productsort_id = get_object_or_404(Productsort,
id=self.kwargs['productsort_id'])
        form.instance.productsort_id = self.productsort_id
        self.area_id = get_object_or_404(Area,
id=self.kwargs['area_id'])
        form.instance.area_id = self.area_id
        return super().form_valid(form)

class MyAdvertisementsView(LoginRequiredMixin, ListView):
    model = Advertisement
    template_name = 'myadverts.html'
    paginate_by = 10

    def get_queryset(self):
        user = self.request.user
        farmer = Farmer.objects.get(id = user.id)
        adverts = Advertisement.objects.filter(farmer_id =
farmer)

        adverts = adverts.order_by('-started_at')
        return adverts

class OwnAdvertisementView (LoginRequiredMixin, DetailView):
    template_name = 'ownadvert.html'
    model = Advertisement

    def get_context_data(self, **kwargs):
        advdata = super().get_context_data(**kwargs)
        adv = get_object_or_404(Advertisement, id =
self.object.pk)
        prods = adv.advert_productions
        prods = prods.filter(state = True)
        advdata ['productions'] = prods
        ams =
Advertisementmark.objects.filter(advertisement_id__id = adv.id)
        advdata ['marks'] = ams

```

```

        advdata ['marksquant'] = ams.count()
        ams =
Preorder.objects.filter(production_id__advertisement_id__id = adv.id)
        advdata ['preordquant'] = ams.count()
        return advdata

class NewProductionView(LoginRequiredMixin, CreateView):
    model = Production
    fields = ['advertisement_id', 'prod_date', 'quantity',
'unit', 'unit_price', 'state', 'is_preordered']
    template_name = 'newproduction.html'

    def get_success_url(self):
        return reverse_lazy('production-new', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        self.advertisement_id =
get_object_or_404(Advertisement, id=self.kwargs['advertisement_id'])
        kwargs['advertisement_id'] = self.advertisement_id
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.advertisement_id =
get_object_or_404(Advertisement, id=self.kwargs['advertisement_id'])
        form.instance.advertisement_id = self.advertisement_id
        return super().form_valid(form)

class UpdateProductionView(LoginRequiredMixin, UpdateView):
    model = Production
    fields = ['advertisement_id', 'prod_date', 'quantity',
'unit', 'unit_price', 'state', 'is_preordered']
    template_name = 'updateproduction.html'

    def get_success_url(self):
        return reverse_lazy('production-update', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        self.advertisement_id =
get_object_or_404(Advertisement, id=self.kwargs['advertisement_id'])
        kwargs['advertisement_id'] = self.advertisement_id
        return super().get_context_data(**kwargs)

    def form_valid(self, form):

```

```

        self.advertisement_id =
get_object_or_404(Advertisement, id=self.kwargs['advertisement_id'])
        form.instance.advertisement_id = self.advertisement_id
        return super().form_valid(form)

class ProductionView(LoginRequiredMixin, DetailView):
    template_name = 'product.html'
    model = Production

    def get_context_data(self, **kwargs):
        proddata = super().get_context_data(**kwargs)
        prod = get_object_or_404(Production, id =
self.object.pk)
        prods = prod.production_preorders
        proddata ['preorders'] = prods
        proddata ['preorderquant'] = prods.count()
        return proddata

class MyProductionsView(LoginRequiredMixin, ListView):
    model = Production
    template_name = 'myprods.html'
    paginate_by = 10

    def get_queryset(self):
        user = self.request.user
        farmer = Farmer.objects.get(id = user.id)
        productions =
Production.objects.filter(production_id__advertisement_id__farmer_id =
farmer)

        productions = productions.order_by('-prod_date')
        return productions

class AdvertisementProductionsView(ListView):
    model = Production
    template_name = 'advprods.html'
    paginate_by = 10

    def get_queryset(self):
        adv = self.request.GET.get('advertisement')
        productions =
Production.objects.filter(advertisement_id = adv)
        productions = productions.order_by('-prod_date')
        return productions

    def get_context_data(self, **kwargs):

```

```

        advdata = super(MyPreordersView,
self).get_context_data(**kwargs)
        advp = self.request.GET.get('advertisement')
        advdata ['advertisement'] = advp
        return advdata

```

```

class NewPreorderView(LoginRequiredMixin, CreateView):
    model = Preorder
    fields = ['production_id', 'department_id', 'needed_at',
'quantity']
    template_name = 'newpreorder.html'

```

```

    def get_success_url(self):
        return reverse_lazy('preorder-new', kwargs={'pk':
self.object.pk})

```

```

    def get_context_data(self, **kwargs):
        self.production_id = get_object_or_404(Production,
id=self.kwargs['production_id'])
        kwargs['production_id'] = self.production_id
        self.department_id = get_object_or_404(Department,
id=self.kwargs['department_id'])
        kwargs['department_id'] = self.department_id
        return super().get_context_data(**kwargs)

```

```

    def form_valid(self, form):
        self.production_id = get_object_or_404(Production,
id=self.kwargs['production_id'])
        form.instance.production_id = self.production_id
        self.department_id = get_object_or_404(Department,
id=self.kwargs['department_id'])
        form.instance.department_id = self.department_id
        return super().form_valid(form)

```

```

class UpdatePreorderView(LoginRequiredMixin, UpdateView):
    model = Preorder
    fields = ['production_id', 'department_id', 'needed_at',
'quantity', 'claim', 'state']
    template_name = 'updatepreorder.html'

```

```

    def get_success_url(self):
        return reverse_lazy('preorder-update', kwargs={'pk':
self.object.pk})

```

```

    def get_context_data(self, **kwargs):

```

```

        self.production_id = get_object_or_404(Production,
id=self.kwargs['production_id'])
        kwargs['production_id'] = self.production_id
        self.department_id = get_object_or_404(Department,
id=self.kwargs['department_id'])
        kwargs['department_id'] = self.department_id
        return super().get_context_data(**kwargs)

```

```

    def form_valid(self, form):
        self.production_id = get_object_or_404(Production,
id=self.kwargs['production_id'])
        form.instance.production_id = self.production_id
        self.department_id = get_object_or_404(Department,
id=self.kwargs['department_id'])
        form.instance.department_id = self.department_id
        buy = Buyer.objects.get(id = user.id)
        form.instance.buyer_id = buy
        return super().form_valid(form)

```

```

class FarmerPreordersView(LoginRequiredMixin, ListView):
    model = Preorder
    template_name = 'farmerpreorders.html'
    paginate_by = 10

    def get_queryset(self):
        user = self.request.user
        statep = self.request.GET.get('state')
        farmer = Farmer.objects.get(id = user.id)
        preorders =
Preorder.objects.filter(production_id__advertisement_id__farmer_id =
farmer)

        preorders = preorders.filter(state = statep)
        preorders = preorders.order_by('-created_at')
        return preorders

```

```

    def get_context_data(self, **kwargs):
        preodata = super(MyPreordersView,
self).get_context_data(**kwargs)
        statep = self.request.GET.get('state')
        preodata ['state'] = statep
        return preodata

```

```

class MyPreordersView(LoginRequiredMixin, ListView):
    model = Preorder
    template_name = 'mypreorders.html'

```

```

paginate_by = 10

def get_queryset(self):
    statep = self.request.GET.get('state')
    user = self.request.user
    buy = Buyer.objects.get(id = user.id)
    preorders = Preorder.objects.filter(buyer_id = buy)
    if statep:
        preorders = preorders.filter(state = statep)
    preorders = preorders.order_by('-created_at')
    return preorders

def get_context_data(self, **kwargs):
    preodata = super(MyPreordersView,
self).get_context_data(**kwargs)
    statep = self.request.GET.get('state')
    preodata ['state'] = statep
    return preodata

class NewAdvertisementmarkView(LoginRequiredMixin, CreateView):
    model = Advertisementmark
    fields = ['advertisement_id', 'mark', 'descrip']
    template_name = 'newadvmark.html'

    def get_success_url(self):
        return reverse_lazy('advmark-new', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        self.advertisement_id =
get_object_or_404(Advertisementmark,
id=self.kwargs['advertisement_id'])
        kwargs['advertisement_id'] = self.advertisement_id
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.advertisement_id =
get_object_or_404(Advertisementmark,
id=self.kwargs['advertisement_id'])
        form.instance.advertisement_id = self.advertisement_id
        buy = Buyer.objects.get(id = user.id)
        form.instance.buyer_id = buy
        return super().form_valid(form)

```

```

class UpdateAdvertisementmarkView(LoginRequiredMixin,
UpdateView):
    model = Advertisementmark
    fields = ['advertisement_id', 'mark', 'descrip']
    template_name = 'updateadvmark.html'

    def get_success_url(self):
        return reverse_lazy('advmark-update', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        self.advertisement_id =
get_object_or_404(Advertisementmark,
id=self.kwargs['advertisement_id'])
        kwargs['advertisement_id'] = self.advertisement_id
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.advertisement_id =
get_object_or_404(UpdatePreorderView,
id=self.kwargs['advertisement_id'])
        form.instance.advertisement_id = self.advertisement_id
        return super().form_valid(form)

class ProductionSearchView(ListView):
    model = Production
    template_name = 'prodsearch.html'
    paginate_by = 10

    def get_queryset(self):
        prodsort = self.request.GET.get('prodsort')
        proddate1 = self.request.GET.get('min_proddate')
        proddate1 = datetime.strptime(proddate1, '%d/%m/%y')
        proddate2 = self.request.GET.get('max_proddate')
        proddate2 = datetime.strptime(proddate2, '%d/%m/%y')
        regionp = self.request.GET.get('region')
        prods = Production.objects.
filter(advertisement_id__productsort_id__id = prodsort)
        if proddate2:
            prods = prods.filter(prod_date__lte = proddate2)
        prods = prods.filter(state = True)
        if proddate1:
            prods = prods.filter(prod_date__gte = proddate1)
        if regionp:

```

```

        prods =
prods.filter(advertisement_id__farmer_id__department_id__area_id__region_id__id = regionp)
        prods = prods.order_by('-advertisement_id__mark')
        return prods

```

```

    def get_context_data(self, **kwargs):
        prods = super(ProductionSearchView,
self).get_context_data(**kwargs)
        prodsort = self.request.GET.get('prodsort')
        proddate1 = self.request.GET.get('min_proddate')
        proddate2 = self.request.GET.get('max_proddate')
        regionp = self.request.GET.get('region')
        prods ['prodsort'] = prodsort
        prods ['min_proddate'] = proddate1
        prods ['max_proddate'] = proddate2
        prods ['region'] = regionp
        return prods

```

```

class AdvertisementSearchView(ListView):

```

```

    model = Advertisement
    template_name = 'advertsearch.html'
    paginate_by = 10

```

```

    def get_queryset(self):
        prodsort = self.request.GET.get('prodsort')
        proddate1 = self.request.GET.get('min_proddate')
        proddate1 = datetime.strptime(proddate1, '%d.%m.%y')
        proddate2 = self.request.GET.get('max_proddate')
        proddate2 = datetime.strptime(proddate2, '%d.%m.%y')
        regionp = self.request.GET.get('region')
        areap = self.request.GET.get('area')
        mark1 = self.request.GET.get('min_mark')
        mark2 = self.request.GET.get('max_mark')
        prodquant = self.request.GET.get('prod_quant')
        advs =

```

```

Advertisement.objects.filter(Q(productsort_id__name__contains =
prodsort) | Q(productsort_id__product_id__name__contains = prodsort))
        if mark2:
            advs = advs.filter(mark__lte = mark2)
        if areap:
            advs = advs.filter(area_id__id = areap)
        if regionp:
            advs = advs.filter(area_id__region_id__id =
regionp)

```

```

        if mark1:
            advs = advs.filter(mark__gte = mark1)
            advs = advs.order_by('-mark')
            for adv in advs:
                prods =
Production.objects.filter(advertisement_id = adv)
                prods = prods.filter(prod_date__gte = proddate1)
                prods = prods.filter(prod_date__lte = proddate2)
                prods = prods.filter(left_quantity__gte =
prodquant)

                if not prods:
                    advs.get(id = adv.id).delete()
            return advs

    def get_context_data(self, **kwargs):
        advs = super(AdvertisementSearchView,
self).get_context_data(**kwargs)
        prodsort = self.request.GET.get('prodsort')
        proddate1 = self.request.GET.get('min_proddate')
        proddate2 = self.request.GET.get('max_proddate')
        regionp = self.request.GET.get('region')
        areap = self.request.GET.get('area')
        mark1 = self.request.GET.get('min_mark')
        mark2 = self.request.GET.get('max_mark')
        prodquant = self.request.GET.get('prod_quant')
        advs ['prodsort'] = prodsort
        advs ['min_proddate'] = proddate1
        advs ['max_proddate'] = proddate2
        advs ['region'] = regionp
        advs ['area'] = areap
        advs ['min_mark'] = mark1
        advs ['max_mark'] = mark2
        advs ['prod_quant'] = prodquant
        return advs

class FarmerPopularAreasView(LoginRequiredMixin, ListView):
    model = Preorder
    template_name = 'farmerpopareas.html'
    paginate_by = 10

    def get_queryset(self):
        user = self.request.user
        farmer = Farmer.objects.get(id = user.id)

```

```

        preorders =
Preorder.objects.filter(production_id__advertisement_id__farmer_id =
farmer)

```

```

        areas = []
        for preord in preorders:
            ar = preord.department_id.area_id
            for (k, a) in areas:
                if ar in a:
                    areas[k] = (ar, a[1]+1)
                else:
                    areas.append((ar,1))
        areas = sorted(areas, key=lambda tup: tup[1])
        return areas

```

```

class MyAdvertisementmarksView(LoginRequiredMixin, ListView):
    model = Advertisementmark
    template_name = 'myadvertmarks.html'
    paginate_by = 10

    def get_queryset(self):
        user = self.request.user
        buy = Buyer.objects.get(id = user.id)
        advm = Advertisementmark.objects.filter(buyer_id =
buy)

        advm = advm.order_by('-created_at')
        return advm

```

```

class MyMarksView(LoginRequiredMixin, ListView):
    model = Advertisementmark
    template_name = 'myadvmarks.html'
    paginate_by = 10

    def get_queryset(self):
        user = self.request.user
        manf = Farmer.objects.get(id = user.id)
        advm =
Advertisementmark.objects.filter(advertisement_id__farmer_id = manf)
        advm = advm.order_by('-created_at')
        return advm

```

```

class FarmerPopularProdsortsView(LoginRequiredMixin, ListView):
    model = Preorder
    template_name = 'farmerpopprodsorts.html'
    paginate_by = 10

```

```

    def get_queryset(self):
        user = self.request.user
        farmer = Farmer.objects.get(id = user.id)
        preorders =
Preorder.objects.filter(production_id__advertisement_id__farmer_id =
farmer)

        prodsorts = []
        for preord in preorders:
            ps =
preord.production_id.advertisement_id.productsort_id
            for (k, a) in prodsorts:
                if ps in a:
                    prodsorts [k] = (ps, a[1] +
preord.quantity)
                else:
                    prodsorts.append((ps,
preord.quantity))
        prodsorts = sorted(prodsorts, key=lambda tup: tup[1])
        return prodsorts

```

```

class NewBuyermarkView(LoginRequiredMixin, CreateView):
    model = Buyermark
    fields = ['buyer_id', 'mark', 'descrip']
    template_name = 'newbuymark.html'

    def get_success_url(self):
        return reverse_lazy('buymark-new', kwargs={'pk':
self.object.pk})

```

```

    def get_context_data(self, **kwargs):
        self.buyer_id = get_object_or_404(Buyer,
id=self.kwargs['buyer_id'])
        kwargs['buyer_id'] = self.buyer_id
        return super().get_context_data(**kwargs)

```

```

    def form_valid(self, form):
        self.buyer_id = get_object_or_404(Buyer,
id=self.kwargs['buyer_id'])
        form.instance.buyer_id = self.buyer_id
        manf = Farmer.objects.get(id = user.id)
        form.instance.buyer_id = manf
        return super().form_valid(form)

```

```

class UpdateBuyermarkView(LoginRequiredMixin, UpdateView):
    model = Buyermark

```

```

fields = ['buyer_id', 'mark', 'descrip']
template_name = 'updateadvmark.html'
def get_success_url(self):
    return reverse_lazy('buymark-update', kwargs={'pk':
self.object.pk})

```

```

def get_context_data(self, **kwargs):
    self.buyer_id = get_object_or_404(Buyer,
id=self.kwargs['buyer_id'])
    kwargs['buyer_id'] = self.buyer_id
    return super().get_context_data(**kwargs)

```

```

def form_valid(self, form):
    self.buyer_id = get_object_or_404(Buyer,
id=self.kwargs['buyer_id'])
    form.instance.buyer_id = self.buyer_id
    manf = Farmer.objects.get(id = user.id)
    form.instance.buyer_id = manf
    return super().form_valid(form)

```

```

class MyBuyermarksView(LoginRequiredMixin, ListView):
    model = Buyermark
    template_name = 'mybuyermarks.html'
    paginate_by = 10

    def get_queryset(self):
        user = self.request.user
        farm = Farmer.objects.get(id = user.id)
        advm = Buyermark.objects.filter(farmer_id = farm)
        advm = advm.order_by('-created_at')
        return advm

```

```

class MyBuyMarksView(LoginRequiredMixin, ListView):
    model = Buyermark
    template_name = 'mybuymarks.html'
    paginate_by = 10

    def get_queryset(self):
        user = self.request.user
        buy = Buyer.objects.get(id = user.id)
        advm = Buyermark.objects.filter(buyer_id__id = buy)
        advm = advm.order_by('-created_at')
        return advm

```

```

class NewLetterView(LoginRequiredMixin, CreateView):

```

```

model = Letter
fields = ['farmer_id', 'buyer_id', 'direction', 'descrip']
template_name = 'newletter.html'

def get_success_url(self):
    return reverse_lazy('letter-new', kwargs={'pk':
self.object.pk})

def get_context_data(self, **kwargs):
    if self.kwargs['direction']:
        self.user_id = get_object_or_404(Buyer,
id=self.kwargs['addruser'])
        kwargs['addruser'] = self.user_id
    else:
        self.user_id = get_object_or_404(Farmer,
id=self.kwargs['addruser'])
        kwargs['addruser'] = self.user_id
    return super().get_context_data(**kwargs)

def form_valid(self, form):
    if self.kwargs['direction']:
        self.user_id = get_object_or_404(Buyer,
id=self.kwargs['addruser'])
        form.buyer_id = self.user_id
        manf = Farmer.objects.get(id = user.id)
        form.instance.farmer_id = manf
    else:
        self.user_id = get_object_or_404(Farmer,
id=self.kwargs['addruser'])
        form.farmer_id = self.user_id
        buy = Buyer.objects.get(id = user.id)
        form.instance.buyer_id = buy
    return super().form_valid(form)

class BuyerView (DetailView):
    template_name = 'bprofile.html'
    model = Buyer

    def get_context_data(self, **kwargs):
        proddata = super().get_context_data(**kwargs)
        buy = get_object_or_404(Buyer, id = self.object.pk)
        marks = buy.buyer_marks
        proddata ['marks'] = marks
        proddata ['marksquant'] = marks.count()
        return proddata

```

```

class FarmerView (DetailView):
    template_name = 'mprofile.html'
    model = Farmer

class LettersView(ListView):
    model = Letter
    template_name = 'letters.html'
    paginate_by = 10

    def get_queryset(self):
        userid = self.request.GET.get('user_id')
        letters = Letter.objects.filter(Q(farmer_id__id =
userid & buyer_id__id = user.id) | Q(farmer_id__id = user.id &
buyer_id__id = userid))
        letters = letters.order_by('id')
        return letters

    def get_context_data(self, **kwargs):
        advdata = super(LettersView,
self).get_context_data(**kwargs)
        userid = self.request.GET.get('user_id')
        advdata ['user_id'] = userid
        return advdata

class AreaFarmersView(ListView):
    model = Farmer
    template_name = 'farmers.html'

    def get_queryset(self):
        areaid = self.request.GET.get('area_id')
        farmers =
Farmer.objects.filter(department_id__area_id__id = areaid)
        return farmers

    def get_context_data(self, **kwargs):
        advdata = super(AreaFarmersView,
self).get_context_data(**kwargs)
        areaid = self.request.GET.get('area_id')
        advdata ['area_id'] = areaid
        return advdata

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»

Дипломна кваліфікаційна робота

Програмне забезпечення пошуку сезонних продуктів

Виконав
ст. гр. КНТ-129сп

Логвиненко Д.С.

Керівник
к.т.н., доц.

Скрупський С.Ю.

2022

Рисунок В.1 – Слайд № 1

Об'єкт роботи – процес продажу сезонних продуктів.

Предмет роботи – програмне забезпечення пошуку сезонних продуктів.

Мета роботи – розробити програмне забезпечення пошуку сезонних продуктів для підтримки прямої взаємодії між фермерами і покупцями.

Рисунок В.2 – Слайд № 2

Завдання дипломної кваліфікаційної роботи

- дослідження проблеми продажу сезонних продуктів;
- визначення засобів розробки;
- витягання вимог до програми;
- проектування програми пошуку сезонних продуктів;
- реалізація програми.

► 3

Рисунок В.3 – Слайд № 3

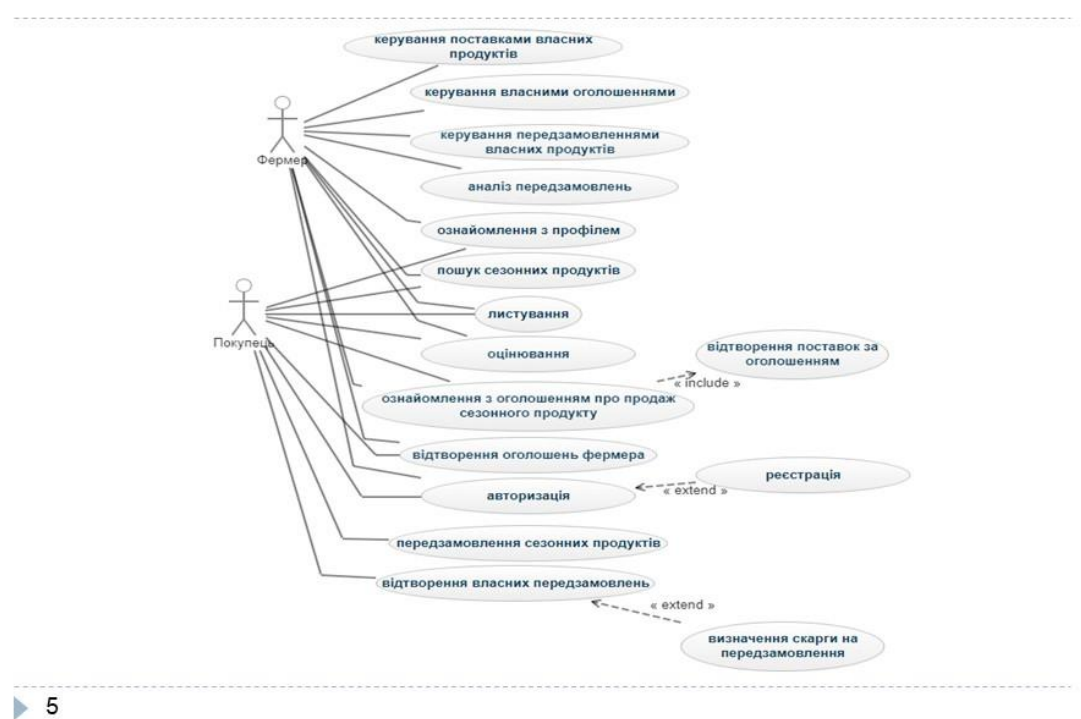
Огляд програмних аналогів

Показник	Seasonalfoodguide.org	Sharingfarm.ca	Власна розробка
Замовлення сезонних продуктів	Ні	Так	Так
Інформація про сезонні продукти на фермі	Ні	Так	Так
Інформація про сезонні продукти	Так	Так	Так
Доступ різних фермерів	Ні	Ні	Так

► 4

Рисунок В.4 – Слайд № 4

Діаграма прецедентів



► 5

Рисунок В.5 – Слайд № 5

Вибір мови програмування для розробки програми

Показник	Java	Python
Менший обсяг коду	Ні	Так
Менші витрати часу на написання коду	Ні	Так
Система типізації динамічна	Ні	Так
Менший час виконання	Так	Ні
Реалізація вебсистем	Так	Так

► 6

Рисунок В.6 – Слайд № 6

Схема бази даних

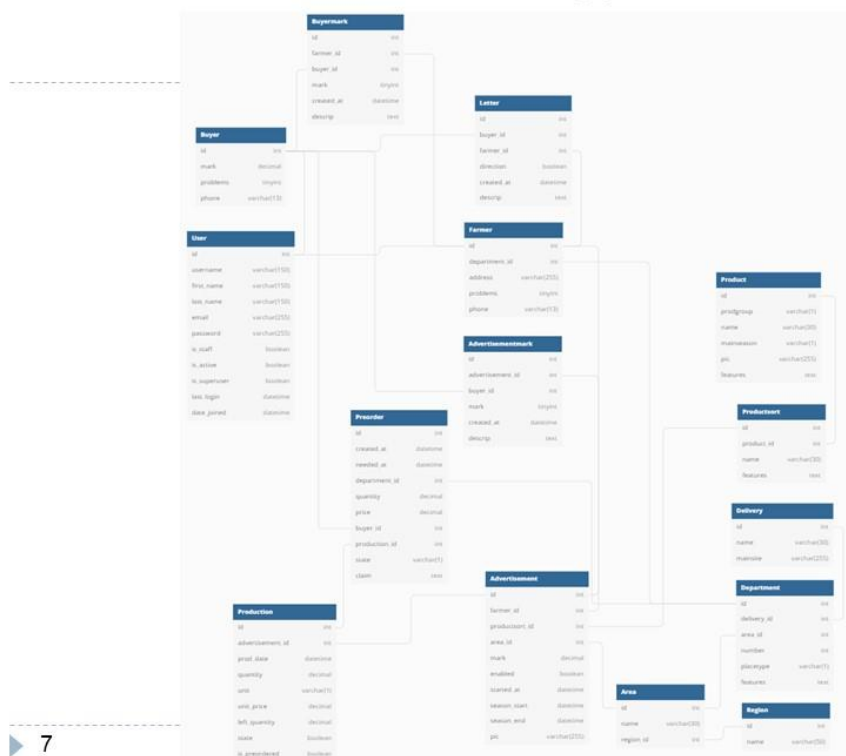


Рисунок В.7 – Слайд № 7

Схема функціонування програми

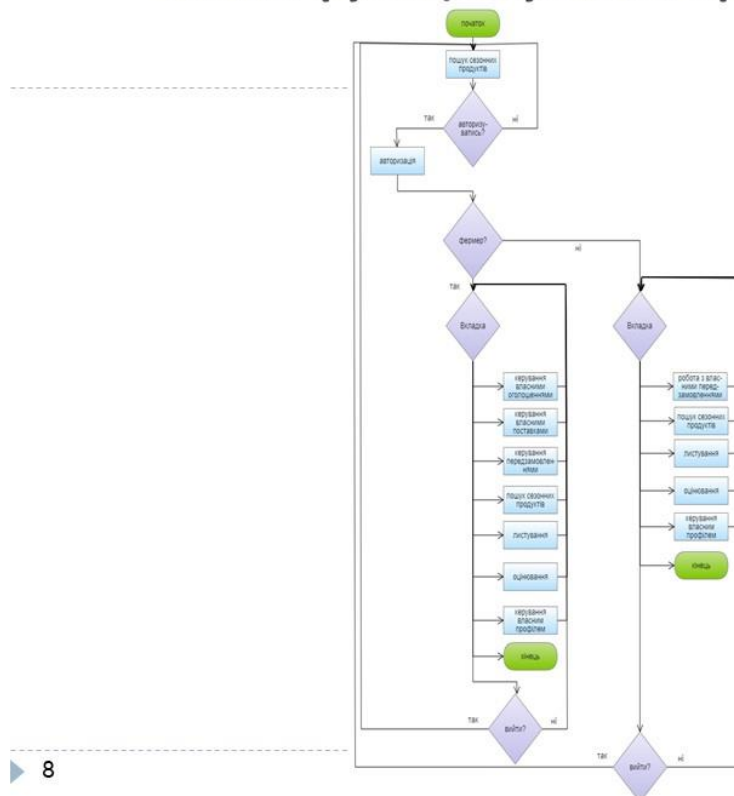


Рисунок В.8 – Слайд № 8

Відтворення списку власних оголошень

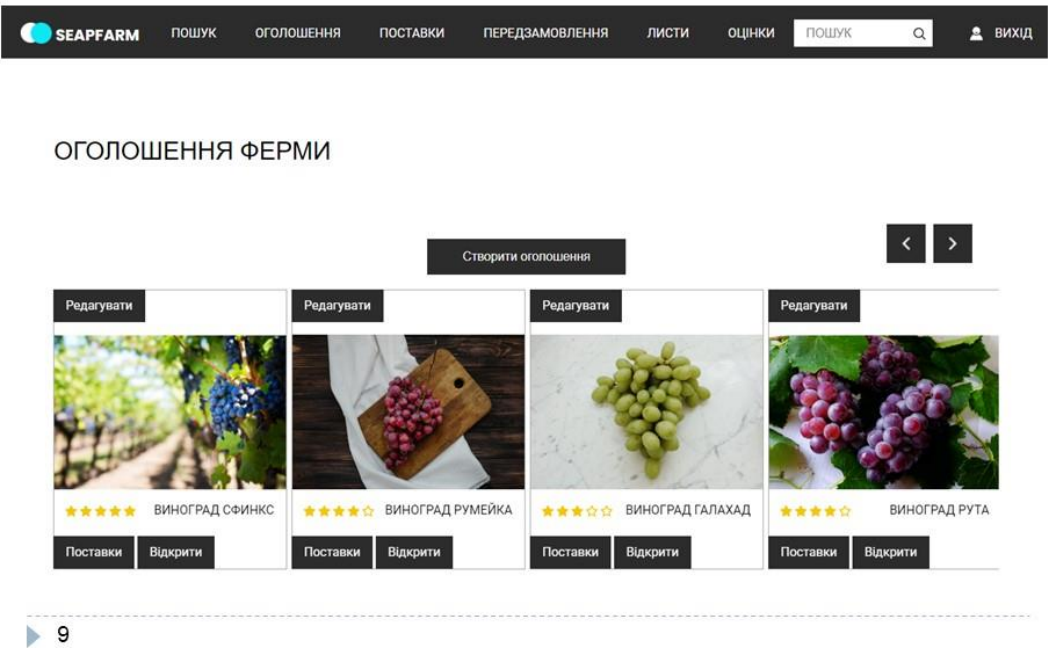


Рисунок В.9 – Слайд № 9

Пошук сезонних продуктів

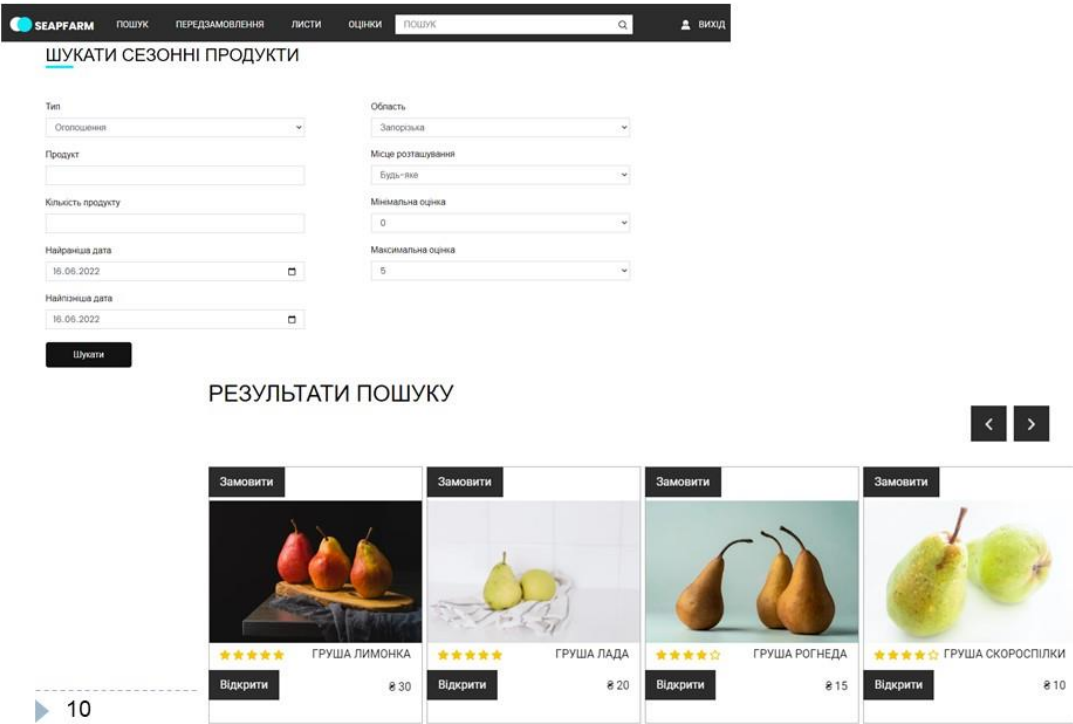


Рисунок В.10 – Слайд № 10

Створення замовлення



ПОШУК

ПЕРЕДЗАМОВЛЕННЯ

ЛИСТИ

ОЦІНКИ

ПОШУК



ВИХІД

СТВОРИТИ ЗАМОВЛЕННЯ

Бажана дата отримання

16.06.2022

Кількість продукту, кг

3

Служба доставки

Нова пошта

Місце доставки

Запоріжжя

Відділення

Відділення № 11

Вартість: 90 грн.

Передзамовити

Груша Лимонка

Лимонарій (Запоріжжя)



► 11

Рисунок В.11 – Слайд № 11

Висновки

Мету розробити програмне забезпечення пошуку сезонних продуктів для підтримки прямої взаємодії між фермерами і покупцями було досягнуто в роботі.

Загальну проблему продажу сезонних продуктів спочатку було проаналізовано, на основі чого сформовано функціональні вимоги, які було перетворено на сценарії роботи покупців і фермерів. Далі сценарії було використано при побудові схеми бази даних, за якою було розроблено моделі вебсистеми, визначено послідовність роботи програми над виконанням описаних функцій, структуровано вебсистему з визначенням вебсторінок з відповідними шаблонами і представленнями, і реалізовано програму в цілому. Розроблені програмні одиниці було описано разом з логікою їх роботи.

► 12

Рисунок В.12 – Слайд № 12