

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
ПРОГНОЗУВАННЯ ПІД ЧАС ПОСТАЧАННЯ ЖИВИХ КВІТІВ
RESEARCH AND SOFTWARE IMPLEMENTATION OF
METHODS FOR FORECASTING THE FRESH FLOWERS SUPPLY

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

Освітня програма (спеціалізація)
Системи штучного інтелекту

МЕРТВІЩЕВ Д.В.

(ПРІЗВИЩЕ та ініціали)

Керівник ЛЬОВКІН В.М.

(ПРІЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРІЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

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

ЗАТВЕРДЖУЮ

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

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

МЕРТВІЩЕВА Дениса Валерійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів прогнозування під час постачання живих квітів. Research and Software Implementation of Methods for Forecasting the Fresh Flowers Supply

керівник проєкту (роботи) к.т.н., доцент ЛЬОВКІН Валерій Миколайович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 7 ” жовтня 2024 року № 398

2. Строк подання студентом проєкту (роботи) 01 грудня 2024 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Проєктування програмного забезпечення. 4. Основні рішення щодо реалізації програмного забезпечення. 5. Експлуатація, тестування та експериментальне дослідження програми.

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

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	ЛЬОВКІН В.М., доцент		
Нормоконтролер	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання “ 08 ” вересня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка)

(підпис)

Денис МЕРТВІЩЕВ

(Імя ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Валерій ЛЬОВКІН

(Імя ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
101 с., 4 табл., 26 рис., 3 дод., 11 джерел.

ЖИВІ КВІТИ, БУКЕТИ, ЗАДАЧА ПРОГНОЗУВАННЯ, АНСАМБЛІ
ДЕРЕВ РІШЕНЬ, МАШИННЕ НАВЧАННЯ, ВЕБЗАСТОСУНОК.

Об'єкт роботи – процеси постачання і продажу живих квітів. Предмет роботи – методи прогнозування під час постачання живих квітів. Мета роботи – розробити програмне забезпечення продажу живих квітів для підвищення зручності роботи користувачів на основі інтеграції прогнозування вартості постачання квітів.

Матеріали, методи та технічні засоби: мова Python з засобами веброзробки, включаючи фреймворк Django, бібліотеку Bootstrap, мови HTML, CSS, JavaScript, система керування базами даних PostgreSQL.

Результати. Вебзастосунок продажу живих квітів дозволяє виконувати прогнозування вартості постачання живих квітів, враховуючи це при формуванні очікуваної вартості продажу квітів для гуртових та індивідуальних покупців, формувати замовлення букетів та сортів квітів, надає засоби для підтримки знаходження потрібних квітів та букетів, дозволяє висловлювати власні оцінки стосовно придбаних живих квітів.

Висновки. Наукова новизна роботи полягає в методі прогнозування вартості постачання живих квітів, який визначає створення і навчання моделі ансамблю дерев рішень на основі методу Random Forest, що дозволяє визначити вартість продажу одиниці квітів на аукціоні. Експериментальне дослідження підтвердило зменшення похибки прогнозування вартості живих квітів порівняно з методом опорних векторів та окремим деревом рішень.

Галузь використання. Продаж та постачання живих квітів зокрема для забезпечення роботи великого магазину.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 101 pages, 4 tables, 26 figures, 3 appendixes, 11 sources.

FRESH FLOWERS, BOUQUETS, FORECASTING PROBLEM, ENSEMBLES OF DECISION TREES, MACHINE LEARNING, WEB APPLICATION.

The object of the work is the process of supply and sale of fresh flowers. The subject of the work is methods of forecasting during the delivery of fresh flowers. The purpose of the work is to develop software for the sale of fresh flowers to improve the user experience based on the integration of forecasting the cost of flower supply.

Materials, methods and technical means: Python language with web development tools, including Django framework, Bootstrap library, HTML languages, CSS, PostgreSQL database management system.

The results. The web application for the sale of fresh flowers allows users to forecast the cost of the supply of fresh flowers, taking this into account when forming the expected cost of selling flowers for wholesale and individual buyers, form orders for bouquets and varieties of flowers, provide tools to support finding the right flowers and bouquets, allows users to express their own assessments of the purchased fresh flowers.

Conclusions. The scientific novelty of the work consists in the method of forecasting the cost of supplying fresh flowers, which determines the creation and training of a model of an ensemble of decision trees based on the Random Forest method, which allows determining the cost of selling a unit of flowers at an auction. An experimental study confirmed the reduction of the forecasting error of the price of fresh flowers compared to the support vector method and a separate decision tree.

Field of use. Sale and supply of fresh flowers, in particular, to ensure the operation of a large store.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз проблеми та постановка завдань дослідження	10
1.1 Огляд вебзастосунків продажу живих квітів	10
1.2 Порівняння аналогів вебзастосунку продажу живих квітів	13
1.3 Висновки за розділом 1	14
2 Матеріали і методи.....	15
2.1 Задача прогнозування вартості постачання живих квітів.....	15
2.2 Метод прогнозування вартості постачання живих квітів	17
2.3 Висновки за розділом 2	20
3 Проєктування програмного забезпечення	21
3.1 Вибір мови та засобів розробки.....	21
3.2 Моделювання роботи з програмним забезпеченням за функціональними вимогами.....	22
3.3 Проєктування структури бази даних.....	26
3.4 Висновки за розділом 3	36
4 Основні рішення щодо реалізації програмного забезпечення	37
4.1 Порядок роботи з основними сутностями програми.....	37
4.2 Рішення з реалізації вебзастосунку	38
4.3 Висновки за розділом 4	54
5 Експлуатація, тестування та експериментальне дослідження програми	55
5.1 Опис програми.....	55
5.2 Експлуатація програми.....	55
5.3 Експериментальне дослідження	60
5.4 Висновки за розділом 5	64
Висновки	65
Перелік джерел посилань	66
Додаток А Технічне завдання	68

Додаток Б Текст програми	73
Додаток В Слайди презентації.....	94

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

CART – Classification and Regression Tree;

MAE – Mean Absolute Error;

MSE – Mean Squared Error;

RMSE – Root Mean Squared Error.

ВСТУП

Робота магазину живих квітів пов'язана з продажем квітів, які при цьому нерідко поставляються з-за меж країни, тобто від постачальників, з якими магазин нерідко безпосередньо не контактує. Одним з таких варіантів є придбання квітів на спеціальних аукціонах закордоном, коли потім квіти доставляються в потрібну країну, а далі продаються в магазинах. Не кожен магазин працює з такими аукціонами, але для великого магазину це є можливим.

Саме на реалізацію підтримки роботи такого магазину і направлена робота програмного застосунку, який потрібно створити в цій роботі. При цьому корисним для користувача в такому випадку є розуміння вартості, за яку він зможе купити квіти, які будуть доставлені в майбутній період часу. Це може дозволити планувати йому свої покупки, а з іншого боку розуміти, якою є очікувана вартість на певні періоди часу. У свою чергу для магазину також важливим є розуміння вартості, щоб планувати далі свою діяльність та надавати відповідну інформацію своїм клієнтам, зокрема гуртовим клієнтам, які виходячи з цього, можуть планувати свою діяльність. Тобто існує цілий ряд взаємопов'язаних процесів. Для цього корисним є наявність функціонально механізму прогнозування вартості постачання живих квітів до магазину. Саме реалізація такого механізму в складі програмного застосунку і є важливим цілісним завданням цієї роботи.

У роботі необхідно розробити програмне забезпечення продажу живих квітів для підвищення зручності роботи користувачів на основі інтеграції прогнозування вартості постачання квітів, таким чином забезпечивши виконання мети роботи.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд вебзастосунків продажу живих квітів

Продаж живих квітів є достатньо активним і актуальним процесом в Україні в цілому, а тому існує багато вебсайтів, де підтримка відповідних процесів забезпечується. Декілька з таких рішень було відібрано для представлення результатів огляду.

Вебсайт Flowers.ua [1] (рис. 1.1) реалізує підтримку продажу букетів живих квітів у різних містах України. При цьому не надається продаж саме звичайних живих квітів (троянд, піонів), а тільки цілісних букетів.



Рисунок 1.1 – Вебсайт Flowers.ua [1]

Flowers.ua має розділи в меню, що включають різні варіанти відображення букетів: акційні букети, нові букети, елітні варіанти, букети квітів за призначенням. За кожним букетом можна обрати певні варіанти для замовлення, а також переглянути відгуки покупців. Також є розділ з останніми відгуками користувачів. Надається поле для пошуку, що фактично є пошуком

за назвою, оскільки можна задати тільки текстове значення, а встановити значення окремих критеріїв неможливо.

Вебсайт ProFlowers.ua [2] (рис. 1.2) містить пропозиції як у вигляді букетів, так і у вигляді окремих, але спеціально представлених квітів як букети (мінімально оформлені). Але придбати саме потрібну кількість квітів (не як оформлений букет) неможливо.

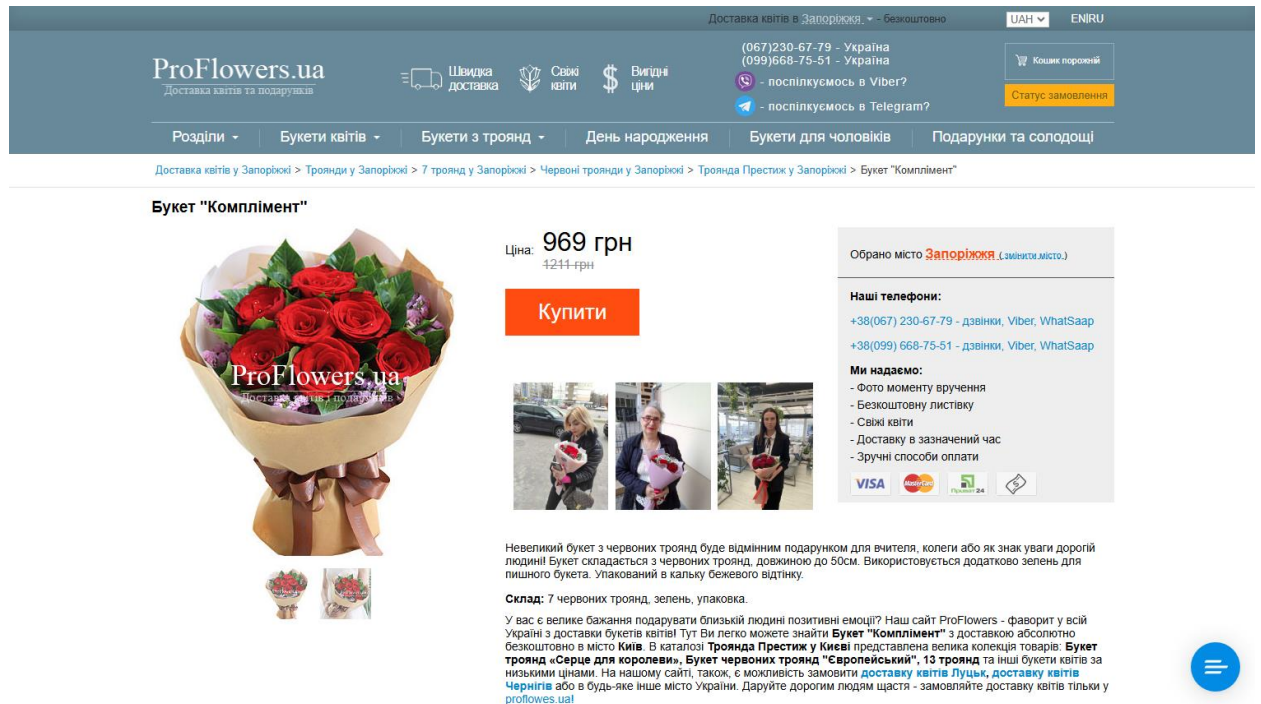


Рисунок 1.2 – Вебсайт ProFlowers.ua [2]

За кожним букетом представлені відгуки. Доступні розділи з різними варіаціями подачі букетів живих квітів. Як такий засіб пошуку букетів живих квітів у програмі відсутній, букети тільки відображаються за певним критерієм, який обрано за відповідним пунктом меню.

Вебсайт Dicentra [3] (рис. 1.3) позиціонується як продаж квітів для великого міста. На вебсайті представлено можливості придбання саме букетів живих квітів. Представлений розділ з доступом до букетів за типом квітів (троянди, півонії тощо), окремо виділено різні варіанти троянд.

Особливістю цього вебсайту як вебзастосунку є доступ до розділу «Квіти», де можна задати різні критерії, зокрема квіти (анемон, анстранція,

антурім тощо), кольори квітів, ціль придбання (привід), кількість квітів у букеті. Також можна обрати спосіб сортування результатів.

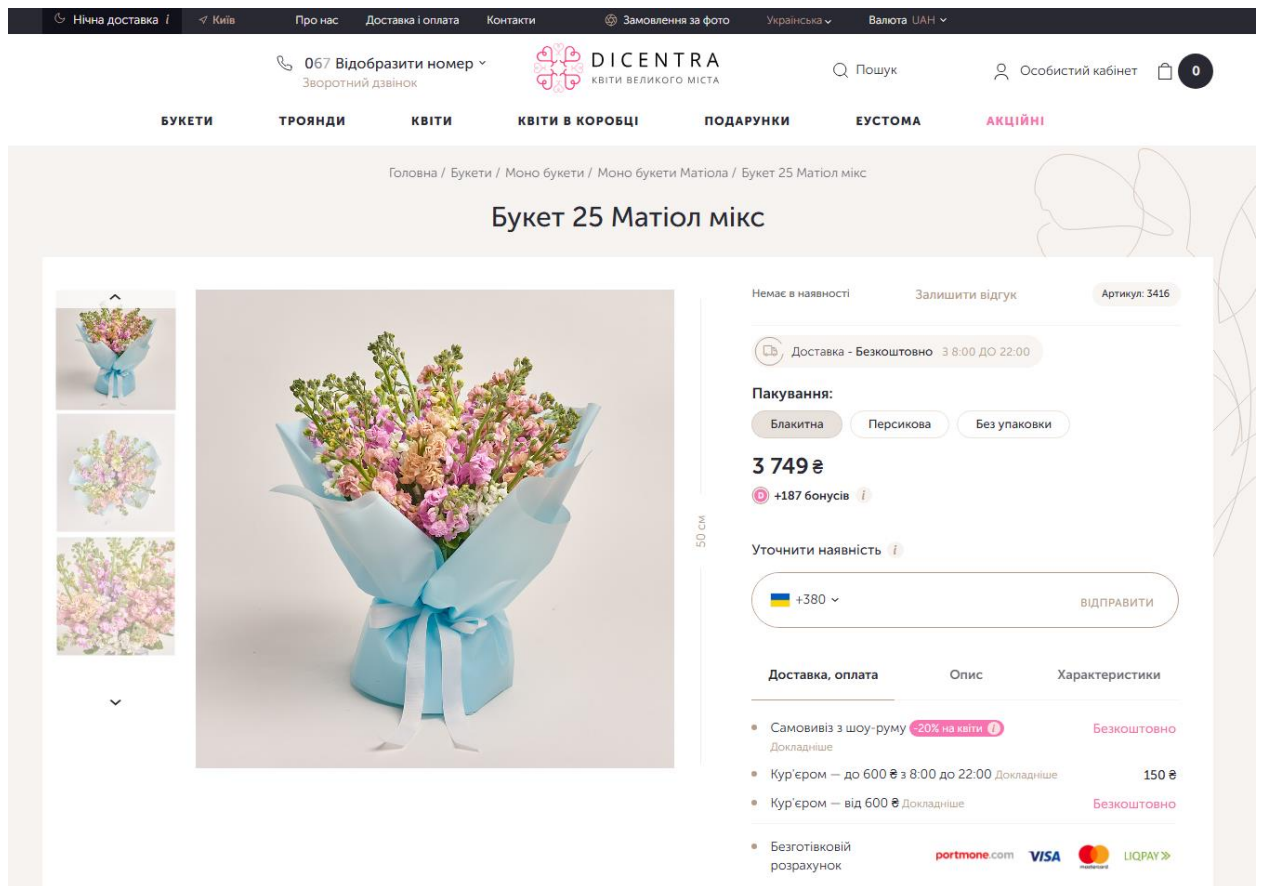


Рисунок 1.3 – Вебсайт Dicentra [3]

За кожним букетом живих квітів можна обрати варіант пакування. Доступні різні варіанти вартості квітів (поточна та наступна на певний період). Проте в цьому способі немає механізму прогнозування. Це фіксована позначка про заплановану зміну вартості. Фактично це маркетинговий інструмент. За букетами представлені відгуки користувачів. Доступлений засіб пошуку живих квітів, але реалізується він також за назвою, тобто представлене тільки текстове поле.

Усі розглянуті вебзастосунки не мають інтегрованих можливостей прогнозування вартості постачання квітів, тому що сам процес постачання не інтегрований фактично в застосунок, тобто робота в програмі реалізується стосовно букетів живих квітів як початкових та кінцевих продуктів.

1.2 Порівняння аналогів вебзастосунку продажу живих квітів

Для порівняння було виділено ті застосунки, які мають основні особливості: Flowers.ua та Dicentra. За визначеними критеріями порівняння відбувалось між ними та запланованою розробкою (табл. 1.1)

Таблиця 1.1 – Порівняння аналогів розробки вебзастосунку продажу живих квітів

Показник	Flowers.ua	Dicentra	Розробка
Перегляд повної інформації про букет	+	+	+
Оцінювання або відгуки про букети живих квітів	+	+	+
Внесення букетів живих квітів до бажань	—	—	+
Багатокритеріальний підбір або пошук живих квітів	—	+	+
Замовлення букетів живих квітів	+	+	+
Замовлення живих квітів	—	—	+
Інтеграція засобів прогнозування вартості живих квітів	—	—	+
Визначення постачань живих квітів	—	—	+

На відміну від існуючих аналогів розробка програми продажу живих квітів об'єднує в собі можливості продажу та постачання живих квітів, інтегруючи в цей процес засоби прогнозування вартості живих квітів, а також надаючи можливість задати заплановані, виконувані постачання. Враховуючи

наявність можливостей постачання, логічним є надання засобів продажу живих квітів через програму, в тому числі оформлення гуртового продажу.

1.3 Висновки за розділом 1

Досягнення мети розроблення програмного забезпечення продажу живих квітів для підвищення зручності роботи користувачів на основі інтеграції прогнозування вартості постачання квітів потрібно виконати через виконання завдань:

- проаналізувати програми продажу живих квітів;
- розробити метод прогнозування вартості постачання живих квітів;
- виконати проєктування програми продажу живих квітів;
- реалізувати програму продажу живих квітів;
- виконати експериментальне дослідження прогнозування вартості постачання живих квітів.

У розділі 1 проаналізовано програми продажу живих квітів, що вже існують, та порівняно їх з розробкою цієї роботи, визначаючи додаткові можливості, які формулюють переваги розробки відносно аналогів.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Задача прогнозування вартості постачання живих квітів

Постачання живих квітів у сучасному світі може здійснюватися далеко не тільки в межах міста або країни, де знаходиться дистриб'ютор. Нерідко квіти доставляються віддалено через велику логістичну мережу, а тому виробники знаходяться у значно віддалених країнах. При цьому за рахунок концентрації таких виробників у певних країнах, масовості галузі взаємодія з ними організована за принципом підтримки справжньої індустрії.

Вираженням цієї підтримки може бути організація аукціонів для продажу живих квітів. Такі аукціони зокрема організовуються в Нідерландах. Тоді на аукціон збираються виробники квітів та потенційні покупці, а вартість продажу квітів визначається на аукціоні. Таким чином формування попиту та пропозиції встановлює баланс у вигляді ціни, визначеної на аукціоні.

При цьому проведення аукціонів надає інформацію, необхідну для планування роботи магазину продажу квітів, дозволяє встановлювати ціну одразу за результатами, а також інформувати клієнтів про можливу вартість у подальшому. Але для того щоб інформувати клієнтів якомога раніше, а також щоб здійснювати планування своєї діяльності з більш достовірною інформацією, потрібно виконувати прогнозування вартості постачання живих квітів. Для цього потрібні відповідні засоби машинного навчання. У такому випадку рішення та надана інформація отримають більш достовірне підґрунтя далі.

У такому випадку важливим стає формулювання задачі прогнозування вартості постачання живих квітів, щоб далі визначити спосіб її розв'язання.

Задача прогнозування вартості постачання живих квітів за своєю основною суттю та характеристиками є задачею регресії. Це задача, в якій необхідно прогнозувати значення вихідної ознаки за значеннями вхідних ознак.

У даному випадку вихідною ознакою є вартість постачання живих квітів. При цьому під вартістю мається на увазі саме сума, за яку можна придбати одиницю постачання, тобто одну квітку. Аукціон призводить до цілісного продажу квітів, тому якщо його обсягів недостатньо, то покупець бере участь в декількох аукціонах. Тоді вони розглядаються як окремі постачання. Відповідно і прогнозування має виконуватися для кожного аукціону окремо.

При цьому додатковим запитанням є те, які саме вхідні ознаки має дана задача. Звичайно, що вхідні ознаки в такому випадку визначаються фактичними характеристиками аукціону. А також початково можуть бути обумовлені тим, як ці дані представляються за результатами проведення аукціону. Виходячи з такої логіки та вибірки [4], і було сформульовано набір вхідних ознак для задачі прогнозування вартості постачання живих квітів.

Вхідними ознаками для вирішення задачі прогнозування вартості постачання живих квітів є:

- кількість учасників аукціону;
- час початку аукціону в секундах;
- відповідність певному варіанту живих квітів;
- здійснення продажу в понеділок;
- здійснення продажу у вівторок;
- здійснення продажу в середу;
- здійснення продажу в четвер.

Фактично ці ознаки характеризують проведення самого аукціону та все те, що на нього в цей момент впливає. Звичайно, що набір ознак може бути і розширений, враховувати різні зовнішні фактори, але якщо робота здійснюється в умовах, які відповідають тим ситуаціям, які раніше ставались, то достатньо охарактеризувати аукціон таким чином. Відповідно дані мають враховувати квіти одного сорту, але можуть представляти декілька варіацій. Тобто однаковий сорт квітів, але різні додаткові характеристики.

Тоді розв'язання задачі прогнозування вартості постачання живих квітів виконується за конкретним заданим сортом живих квітів.

Враховуючи, що дана задача є задачею регресії, то для її розв'язання має виконуватися навчання з учителем. Відповідно необхідна вибірка даних, яка буде характеризувати раніше проведені випадки такого аукціону.

2.2 Метод прогнозування вартості постачання живих квітів

Як було визначено, задача прогнозування вартості постачання живих квітів є задачею регресії. Така задача може мати ряд розв'язків на основі методів машинного навчання.

Найпростішим методом, який може використовуватися для розв'язання задачі регресії, є лінійна регресія. Така модель базується на побудові простої лінійної залежності між вхідною ознакою та вихідною. Однак проблемою застосування на практиці такої моделі є те, що нерідко залежність, яка формується між ознаками, не є лінійною [5]. Ще одним варіантом методів прогнозування є метод опорних векторів [6].

Одним з варіантів, як може моделюватися нелінійна залежність у машинному навчанні, є дерева рішень. Такі дерева рішень нерідко базуються на алгоритмі Classification and Regression Tree (CART), хоча існують і інші подібні алгоритми [7]. Даний алгоритм може застосовуватися як для розв'язання задачі регресії, так і класифікації, що робить його універсальним.

Основна функція алгоритму дерева рішень полягає в тому, щоб розділити набір даних на менші набори, відповідно підмножини загального набору даних створюються для представлення значення будь-якої точки даних, яка пов'язана з постановкою проблеми, у свою чергу поділ набору даних за допомогою цього алгоритму призводить до створення дерева рішень, яке має вузли, які визначаються прийняття певних рішень, і листові вузли [5].

Ще одним варіантом створення моделей для розв'язання задачі регресії, а відповідно і задачі прогнозування вартості постачання живих квітів,

є метод Random Forest [8]. Загалом цей алгоритм є широко використовуваним алгоритмом для розв'язання задачі регресії в машинному навчанні, коли залежність, що вивчається, є нелінійною, при цьому на відміну від дерев рішень (наприклад, CART), Random Forest використовує декілька дерев рішень для прогнозування підсумкового результату, при цьому з заданого початкового набору даних вибираються випадкові точки даних і за допомогою цього алгоритму на основі виключно цих точок будується дерево рішень, потім моделюється декілька дерев рішень, які відповідно мають свою спеціалізацію на основі вибору інших точок даних з початкового набору, а тоді вже кожне з цих дерев використовується для того, щоб передбачити значення будь-якої нової точки даних, при цьому оскільки існує декілька дерев рішень, то і декілька вихідних значень буде передбачено за допомогою алгоритму Random Forest, тоді необхідно знайти середнє значення всіх прогнозованих значень для того екземпляру даних, який було отримано на вхід, щоб обчислити кінцевий результат [5].

У даному методі прогнозування вартості постачання живих квітів саме метод Random Forest використано для створення моделі прогнозування.

Загалом же метод передбачає виконання послідовності етапів.

Спочатку має бути реалізовано підготовку даних у вигляді навчальної вибірки даних, включаючи ряд значень ознак для кожного екземпляру даних. Кожен екземпляр даних представляє проведений аукціон з продажу живих квітів. При цьому усі екземпляри даних стосуються аукціонів з продажу квітів певного сорту. Для роботи з аукціонами за іншим сортом квітів має окремо реалізовуватися метод зі створенням іншої моделі.

В якості ознак для кожного екземпляру даних запропоновано використовувати:

- кількість учасників аукціону;
- час початку аукціону в секундах, що вказується з початку проведення аукціонів у відповідний день;

– відповідність певному варіанту живих квітів: якщо на аукціоні продається декілька основних варіантів сорту живих квітів, то вони визначаються окремими ознаками, тобто за кожним варіантом вводиться окрема ознака, а значення цієї ознаки дорівнює 1, якщо екземпляр даних належить цьому варіанту сорту, та 0 у протилежному випадку (якщо варіантів два, то відповідно дві ознаки та у кожного екземпляра 1 розташована в одній ознаці та 0 у іншій, а якщо 3, то 1 розташовано в одній ознаці, а 0 у всіх двох інших);

– здійснення продажу у відповідний день тижня: в залежності від кількості днів тижня, в які працює аукціон, кількість ознак змінюється, а значення кожної ознаки відповідає 1, якщо аукціон проводився у відповідний день тижня (наприклад, понеділок) та 0 у протилежному випадку.

Після того, як для кожного екземпляру даних визначено набір значень за кожної з відповідних ознак, до екземпляру має бути додано вартість постачання живих квітів, яка була отримана в даному аукціоні. Усі значення вартостей мають бути пронормовані, а тоді пронормовані значення мають бути додані до відповідних екземплярів. Сформована таким чином вибірка використовується для подальшої роботи.

На наступному етапі формується модель для прогнозування вартості постачання живих квітів на основі створення ансамблю дерев рішень за допомогою методу Random Forest. Кожне дерево спеціалізується на певній частині навчальної вибірки даних, а підсумкове рішення про вартість постачання приймається шляхом усереднення значень, отриманих прогнозуванням вартості кожним з дерев ансамблю.

Сформована таким чином модель застосовується для навчання на основі створеної на першому етапі вибірки даних.

Останній етап реалізується при появі кожного нового екземпляру даних, для якого невідома вартість постачання живих квітів, якщо ці живі квіти належать до заданого сорту, що продається на цих аукціонах. Тоді такий аукціон має бути визначений на основі значень вхідних ознак, формуючи

таким чином екземпляр даних, після чого він подається на вхід навченої моделі, а на виході отримується результат прогнозування – вартість постачання одиниці живих квітів. Отримане значення є нормованим, тому в кінці воно має бути перетворене на значення у відповідній валюті, використовуючи початкові максимальне та мінімальні значення у навчальній вибірці.

2.3 Висновки за розділом 2

Наукова новизна роботи полягає в методі прогнозування вартості постачання живих квітів, який визначає створення і навчання моделі ансамблю дерев рішень на основі методу Random Forest, що дозволяє визначити вартість продажу одиниці квітів на аукціоні.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір мови та засобів розробки

Оскільки програма продажу живих квітів в основі має функціональні можливості, які можна віднести до реалізації магазину, то типовим рішенням для реалізації таких можливостей є системи управління вмістом, зокрема Wordpress [9], а альтернативою є фреймворки вебробки, одним з найпотужніших з яких є Django [10] (табл. 3.1).

Таблиця 3.1 – Вибір засобів розробки програми продажу живих квітів

	Wordpress	Django
Тип забезпечуваного рішення	Система управління вмістом	Фреймворк вебробки
Основна мова розробки	PHP	Python
Простота створення стандартних функцій магазину	Вища	Нижча
Простота створення додаткових функцій магазину	Нижча	Вища
Безпечність підсумкового рішення	Нижча	Висока
Рівень підтримки для створення моделей прогнозування відповідною мовою розробки	Нижчий	Високий

Django та Python обрано для реалізації, враховуючи більшу простоту Python для реалізації моделей прогнозування, опрацювання даних, а також

простоту створення додаткових функцій магазину, безпечність підсумкового рішення, що важливо для функціональності такої програми. При цьому реалізація стандартних рішень магазину у Wordpress простіша, але наявність додаткових функцій, можливостей прогнозування потребує створення додаткових рішень, а це вже значно більші витрати (зокрема часові) ніж у Django.

3.2 Моделювання роботи з програмним забезпеченням за функціональними вимогами

Усі користувачі розділяються на дві великі групи.

До першої належать покупці магазину живих квітів, які використовують вебзастосунок і які є або звичайними індивідуальними покупцями, або гуртовими покупцями. Для всіх них доступний однаковий набір сценаріїв, але в залежності від конкретної категорії можуть бути певні обмеження всередині сценаріїв. Це пов'язано з тим, що індивідуальні покупці не можуть замовляти квіти в тій кількості, що гуртові, адже для цього потрібна спеціальна підтримка даного процесу при обслуговуванні.

До другої групи належать представники самого магазину живих квітів.

Основні сценарії для роботи покупців магазину живих квітів:

- відбір потрібного букету живих квітів;
- відбір індивідуальних живих квітів;
- замовлення живих квітів;
- перегляд інформації про своє замовлення живих квітів;
- перегляд інформації про себе;
- додавання букету живих квітів або сортів квітів до бажання;
- перегляд бажаних букетів або сортів живих квітів;
- оцінювання букету або сорту живих квітів;
- перегляд самостійно оцінених букетів або сортів живих квітів;
- перегляд своїх виконаних замовлень живих квітів;

- перегляд інформації про букет живих квітів;
- перегляд інформації про сорт живих квітів;
- авторизація покупця (рис. 3.1).

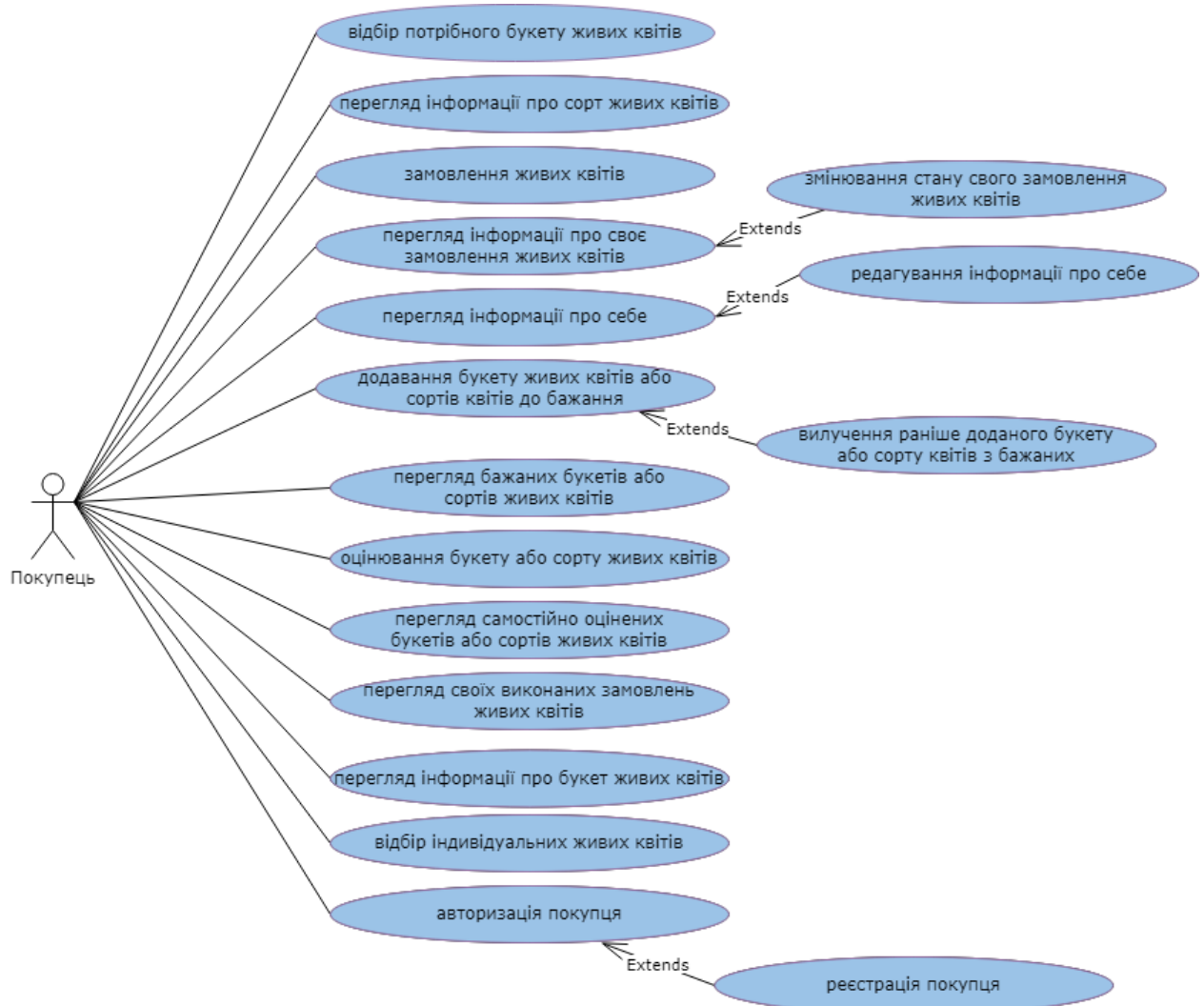


Рисунок 3.1 – Діаграма варіантів використання програми покупцями магазину живих квітів

Альтернативні сценарії цього основного потоку:

- змінювання стану свого замовлення живих квітів, що є альтернативним для перегляду інформації про своє замовлення живих квітів і реалізується тоді, коли створене замовлення ще не було далі опрацьовано представником магазину, а актуальність замовлення зникла;

- редагування інформації про себе, що є альтернативним для перегляду інформації про себе, коли потрібно внести правки;

- вилучення раніше доданого букету або сорту квітів з бажаних, що є альтернативним для додавання букету живих квітів або сортів квітів до бажання, коли покупець вже додав цей букет або цей сорт живих квітів до власних бажань;

- реєстрація покупця, що є альтернативним для авторизації покупця на випадок, коли покупець не має зареєстрованого облікового запису для виконання авторизації.

Основний потік сценаріїв роботи представника магазину живих квітів з програмою:

- замовлення живих квітів;
- перегляд інформації про замовлення живих квітів;
- перегляд інформації про користувача;
- змінювання стану замовлення живих квітів, що не є альтернативним, а саме є основним сценарієм, тому що будь-яке замовлення має бути обов'язково опрацьовано представником магазину живих квітів у подальшому, а це передбачає зміну стану;

- перегляд оцінених користувачем букетів або сортів живих квітів;
- додавання нового постачання живих квітів, що включає сценарій прогнозування вартості постачання живих квітів;

- перегляд запланованих та виконаних постачань живих квітів;
- визначення наявності сортів живих квітів та букетів;
- перегляд замовлень живих квітів у магазині;
- додавання інформації про букет живих квітів;
- пошук замовлення живих квітів;
- додавання сортів живих квітів;
- перегляд інформації про букет живих квітів;
- перегляд інформації про сорт живих квітів;
- пошук серед сортів живих квітів;

– пошук серед букетів живих квітів;
 – авторизація представників магазину живих квітів;
 – реєстрація представників магазину живих квітів: для представника магазину живих квітів це не альтернативний сценарій, а саме основний, тому що цей сценарій має бути доступний як основна функція, адже саме представник магазину може зареєструвати іншого представника магазину, а не він сам себе, як це є у покупця (рис. 3.2).



Рисунок 3.2 – Діаграма варіантів використання програми представниками магазину живих квітів

Альтернативні сценарії цього основного потоку:

- редагування даних постачання живих квітів, що є альтернативним для додавання нового постачання живих квітів, коли таке додавання було реалізовано, а дані необхідно змінити;
- редагування інформації про букет живих квітів, що є альтернативним для додавання інформації про букет живих квітів, коли таке додавання було реалізовано, а дані необхідно змінити;
- редагування інформації про сорт живих квітів, що є альтернативним для додавання сортів живих квітів, коли таке додавання було реалізовано, а дані необхідно змінити.

3.3 Проєктування структури бази даних

Визначення структури бази даних є ключем для реалізації функціональних можливостей, які пов'язані з роботою з даними. Тому використовуючи запланований перелік сценаріїв, було проаналізовано та виділено ті сутності, які існують в цій предметній області. Після чого цю структуру разом зі зв'язками між ними було перетворено на базу даних на основі системи PostgreSQL.

Види живих квітів (тройнди, піони тощо) представлені таблицею Flowerkind зі складом таких полів:

- id – первинний ключ для ідентифікації виду живих квітів;
- kindname – назва виду живих квітів;
- basephoto – фотографія варіації сорту живих квітів, яка використовується для ілюстрації цього виду живих квітів при відображенні такої фотографії на сторінці, відповідно є зовнішнім ключем до таблиці FlowersortvariantPhoto.

Сорти живих квітів представлені таблицею Flowersort зі складом таких полів:

- id – первинний ключ для ідентифікації сорту живих квітів;

– flowerkind – вид живих квітів, до якого належить цей сорт, відповідно є зовнішнім ключем до таблиці Flowerkind;

– sortname – назва сорту живих квітів;

– description – опис особливостей цього сорту живих квітів;

– basephoto – фотографія варіації сорту живих квітів, яка використовується для ілюстрації цього сорту живих квітів при відображенні такої фотографії на сторінці, відповідно є зовнішнім ключем до таблиці FlowersortvariantPhoto.

Варіація сорту живих квітів, яка є позицією в каталогі магазину, представлена таблицею Flowersortvariant зі складом таких полів:

– id – первинний ключ для ідентифікації варіації сорту живих квітів, що може продаватися у магазині;

– flowersort – сорт квітів, варіацією якого є дана, відповідно є зовнішнім ключем до таблиці Flowersort;

– size – розмір квітки у цієї варіації даного сорту живих квітів;

– color – колір квітки у цієї варіації даного сорту живих квітів;

– height – висота (довжина ніжки) цієї варіації даного сорту живих квітів;

– description – опис особливостей цієї варіації (якщо такі особливості присутні);

– basephoto – фотографія цієї варіації сорту живих квітів, яка використовується при відображенні такої фотографії на сторінці, відповідно є зовнішнім ключем до таблиці FlowersortvariantPhoto;

– basesupply – постачання цієї варіації цього сорту живих квітів, з якого в замовленнях використовуються квіти на даний момент, відповідно є зовнішнім ключем до таблиці Flowersortvariantsupply;

– score – середня оцінка, отримана від користувачів.

Фотографії варіацій сортів живих квітів представлені таблицею FlowersortvariantPhoto зі складом таких полів:

– id – первинний ключ для ідентифікації варіації сорту живих квітів;

– `flowersortvariant` – варіація сорту живих квітів, яка показана на фотографії, відповідно є зовнішнім ключем до таблиці `Flowersortvariant`;

– `photo` – дані фотографії (саме зображення).

Пакування, які доступні для конкретної варіації сорту живих квітів, представлені таблицею `FlowersortvariantPackaging` зі складом таких полів:

– `id` – первинний ключ для ідентифікації пакування варіації сорту живих квітів;

– `flowersortvariant` – варіація сорту живих квітів, для якої доступне це пакування, відповідно є зовнішнім ключем до таблиці `Flowersortvariant`;

– `packaging` – пакування, в яке можна оформити цю варіацію живих квітів, відповідно є зовнішнім ключем до таблиці `Packaging`;

– `price` – вартість самого пакування цієї варіації сорту живих квітів.

Постачання варіацій сортів живих квітів представлені таблицею `Flowersortvariantsupply` зі складом таких полів:

– `id` – первинний ключ для ідентифікації постачання варіації сорту живих квітів;

– `flowersortvariant` – варіація сорту живих квітів, постачання якої здійснюється в даному випадку, відповідно є зовнішнім ключем до таблиці `Flowersortvariant`;

– `number` – кількість одиниць живих квітів у цій варіації даного сорту, що закуплена і має бути поставлена для продажу;

– `leftnumber` – кількість одиниць живих квітів у цій варіації даного сорту, яка ще на даний момент не є проданою або замовленою;

– `status` – стан цього постачання цієї варіації живих квітів;

– `plannedprice` – запланована вартість придбання одиниці цієї варіації живих квітів;

– `forecastprice` – прогнозована вартість придбання одиниці цієї варіації живих квітів;

– `fixedprice` – фактична вартість, за якою буде продаватися одиниця цієї варіації живих квітів;

– `addpercentprice` – відсоток націнки на вартість придбання цієї варіації живих квітів у постачальників для формування вартості подальшого продажу магазином;

– `addprice` – додаткова націнка (абсолютне значення) на вартість придбання цієї варіації живих квітів у постачальників для формування вартості подальшого продажу магазином (може представляти собою вартість доставки такої партії живих квітів);

– `bidders` – кількість учасників аукціону з продажу цієї варіації живих квітів;

– `buydatetime` – дата і час проведення аукціону живих квітів;

– `supplydate` – дата, коли має бути поставлено в магазин від постачальника цю партію живих квітів.

Цілі, за якими можуть куплятися букети живих квітів, представлені таблицею `Purpose` зі складом таких полів:

– `id` – первинний ключ для ідентифікації цілі придбання живих квітів;

– `purposename` – назва (формулювання) цілі придбання живих квітів.

Букети живих квітів у магазині представлені таблицею `Bouquet` зі складом таких полів:

– `id` – первинний ключ для ідентифікації букету живих квітів;

– `caption` – назва букету живих квітів;

– `description` – опис букету живих квітів;

– `basephoto` – фотографія цього букету живих квітів, яка використовується при відображенні такої фотографії на сторінці, відповідно є зовнішнім ключем до таблиці `BouquetPhoto`;

– `priceaddpart` – відсоток націнки на сумарну вартість окремих квітів при формуванні вартості букету;

– `fixedaddprice` – додаткова націнка (абсолютне значення) на вартість окремих квітів при формуванні вартості букету;

– `fixedprice` – підсумкова стандартна вартість продажу букету (може задаватися вручну без розрахунку);

– recalculate – позначка про те, чи треба переобчислювати вартість букету при постачанні нових живих квітів, чи потрібну використовувати значення з поля fixedprice постійно;

– currentfixedprice – підсумкова акційна вартість продажу букету;

– maxdaybouquets – максимальна кількість букетів, яка може бути продана на день;

– score – середня оцінка, отримана від користувачів.

Цілі, за якими рекомендовано купувати конкретні букети живих квітів, представлені таблицею BouquetPurpose зі складом таких полів:

– id – первинний ключ для ідентифікації цілі, за якою рекомендовано купувати конкретний букет живих квітів;

– bouquet – букет живих квітів, для купівлі якого рекомендована ця ціль, відповідно є зовнішнім ключем до таблиці Bouquet;

– purpose – ціль, за якою рекомендовано купляти цей букет живих квітів, відповідно є зовнішнім ключем до таблиці Purpose.

Можливі конкретні пакування конкретних букетів живих квітів представлені таблицею BouquetPackaging зі складом таких полів:

– id – первинний ключ для ідентифікації пакування букету живих квітів;

– bouquet – букет живих квітів, пакування для якого визначено, відповідно є зовнішнім ключем до таблиці Bouquet;

– packaging – пакування, яке може використовуватися для цього букету живих квітів, відповідно є зовнішнім ключем до таблиці Packaging;

– price – вартість такого пакування;

– photo – дані фотографії (саме зображення) цього пакування букету живих квітів.

Живі квіти, які входять до букету, представлені таблицею BouquetFlower зі складом таких полів:

– id – первинний ключ для ідентифікації складу живих квітів у букеті;

– bouquet – букет живих квітів, до якого входить ця варіація сорту живих квітів, відповідно є зовнішнім ключем до таблиці Bouquet;

– flowersortvariant – варіація сорту живих квітів, що входить до складу цього букету, відповідно є зовнішнім ключем до таблиці Flowersortvariant;

– number – кількість одиниць цієї варіації живих квітів, що використовується в даному букеті;

– priceaddpart – відсоток націнки на одиницю цієї квітки при формуванні вартості букету.

Фотографії букетів живих квітів представлені таблицею BouquetPhoto зі складом таких полів:

– id – первинний ключ для ідентифікації фотографії букету живих квітів;

– bouquet – букет живих квітів, який зображений на фотографії, відповідно є зовнішнім ключем до таблиці Bouquet;

– photo – дані фотографії (саме зображення).

Користувачі програми продажу живих квітів представлені таблицею FlowershopUser зі складом таких полів:

– id – первинний ключ для ідентифікації користувача програми продажу живих квітів;

– djangouser – обліковий запис Django, за яким авторизується в програмі цей користувач;

– fullname – повне ім'я користувача програми продажу живих квітів;

– phone – номер телефону користувача програми продажу живих квітів;

– wholesale – позначка про те, що цей користувач представляє гуртового покупця живих квітів;

– aboutme – опис користувачем програми продажу живих квітів себе;

– delivery – адреса доставки, яка може використовуватися під час формування замовлення живих квітів стандартно (якщо є служба доставки).

Замовлення живих квітів представлені таблицею FlowershopOrder зі складом таких полів:

- id – первинний ключ для ідентифікації замовлення живих квітів;
- flowershopUser – користувач програми продажу живих квітів, який сформував замовлення, відповідно є зовнішнім ключем до таблиці FlowershopUser;
- currentstatus – поточний стан роботи над цим замовленням живих квітів;
- createdat – дата і час, коли було створено це замовлення живих квітів;
- deliveryat – дата і час, коли має бути готове це замовлення живих квітів;
- deliveryad – адреса доставки замовлення букету живих квітів, якщо наявна служба доставки;
- prompt – підказка щодо особливостей цього замовлення живих квітів;
- totalprice – вартість цього замовлення живих квітів.

Історія зміни станів замовлень живих квітів представлена таблицею FlowershopOrderStatus зі складом таких полів:

- id – первинний ключ для ідентифікації запису про зміну станів замовлень живих квітів;
- flowershopuser – користувач, який вніс цю зміну стану замовлення живих квітів;
- statusvalue – значення стану, яке було встановлено;
- changedat – дата і час, коли ця зміна стану замовлення живих квітів була виконана.

Букети живих квітів, що входять до складу замовлення живих квітів, представлені таблицею FlowershopOrderBouquet зі складом таких полів:

- id – первинний ключ для ідентифікації входження букету живих квітів до замовлення;
- flowershoporder – замовлення живих квітів, в яке входить цей букет, відповідно є зовнішнім ключем до таблиці FlowershopOrder;
- bouquet – букет живих квітів, який входить в це замовлення, відповідно є зовнішнім ключем до таблиці Bouquet;

- number – кількість таких букетів живих квітів у замовленні;
- price – вартість одного букету живих квітів.

Варіації сортів живих квітів, що входять до складу замовлення живих квітів, представлені таблицею `FlowershopOrderFlowersortvariant` зі складом таких полів:

- id – первинний ключ для ідентифікації входження варіації сорту живих квітів до замовлення;
- flowershoporder – замовлення живих квітів, в яке входить ця варіація сорту, відповідно є зовнішнім ключем до таблиці `FlowershopOrder`;
- flowersortvariant – варіація сорту живих квітів, яка входить до замовлення, відповідно є зовнішнім ключем до таблиці `Flowersortvariant`;
- number – кількість одиниць квітів цієї варіації даного сорту в замовленні;
- price – вартість одиниці квітів цієї варіації даного сорту в замовленні.

Бажання користувачем варіацій сортів живих квітів представлені таблицею `Flowersortvariantdesire` зі складом таких полів:

- id – первинний ключ для ідентифікації бажання користувачем варіації сорту живих квітів;
- flowersortvariant – варіація сорту живих квітів, яка була збережена користувачем у бажання, відповідно є зовнішнім ключем до таблиці `Flowersortvariant`;
- flowershopuser – користувач програми продажу живих квітів, який зберіг у бажання цю варіацію сорту живих квітів, відповідно є зовнішнім ключем до таблиці `FlowershopUser`.

Бажання користувачем букетів живих квітів представлені таблицею `Bouquetdesire` зі складом таких полів:

- id – первинний ключ для ідентифікації бажання користувачем букету живих квітів;

– flowershopuser – користувач програми продажу живих квітів, який зберіг у бажання цей букет живих квітів, відповідно є зовнішнім ключем до таблиці FlowershopUser;

– bouquet – букет живих квітів, який був збережений у бажання цим користувачем, відповідно є зовнішнім ключем до таблиці Bouquet.

Оцінювання варіацій сортів живих квітів користувачами програми продажу живих квітів представлені таблицею Flowersortvariantscore зі складом таких полів:

– id – первинний ключ для ідентифікації оцінювання варіації сортів живих квітів користувачем програми продажу живих квітів;

– flowershopuser – користувач програми продажу живих квітів, який оцінив цю варіацію сорту живих квітів, відповідно є зовнішнім ключем до таблиці FlowershopUser;

– flowersortvariant – варіація сорту живих квітів, яка була оцінена користувачем програми продажу живих квітів, відповідно є зовнішнім ключем до таблиці Flowersortvariant;

– score – виставлена оцінка;

– scoredat – дата, час, коли оцінка варіації живих квітів була виставлена користувачем програми продажу живих квітів.

Оцінювання букетів живих квітів користувачами програми продажу живих квітів представлені таблицею Bouquetscore зі складом таких полів:

– id – первинний ключ для ідентифікації оцінювання букету живих квітів користувачем програми продажу живих квітів;

– flowershopuser – користувач програми продажу живих квітів, який оцінив цей букет живих квітів, відповідно є зовнішнім ключем до таблиці FlowershopUser;

– bouquet – букет живих квітів, який був оцінений користувачем програми продажу живих квітів, відповідно є зовнішнім ключем до таблиці Bouquet;

– score – виставлена оцінка;

– scoredat – дата, час, коли оцінка букету живих квітів була виставлена користувачем програми продажу живих квітів.

Доступні пакування живих квітів представлені таблицею Packaging зі складом таких полів (рис. 3.3):

- id – первинний ключ для ідентифікації пакування живих квітів;
- packagename – назва пакування живих квітів;
- packagedescription – додатковий опис особливостей цього пакування;
- photo – дані загальної фотографії цього пакування (саме зображення).

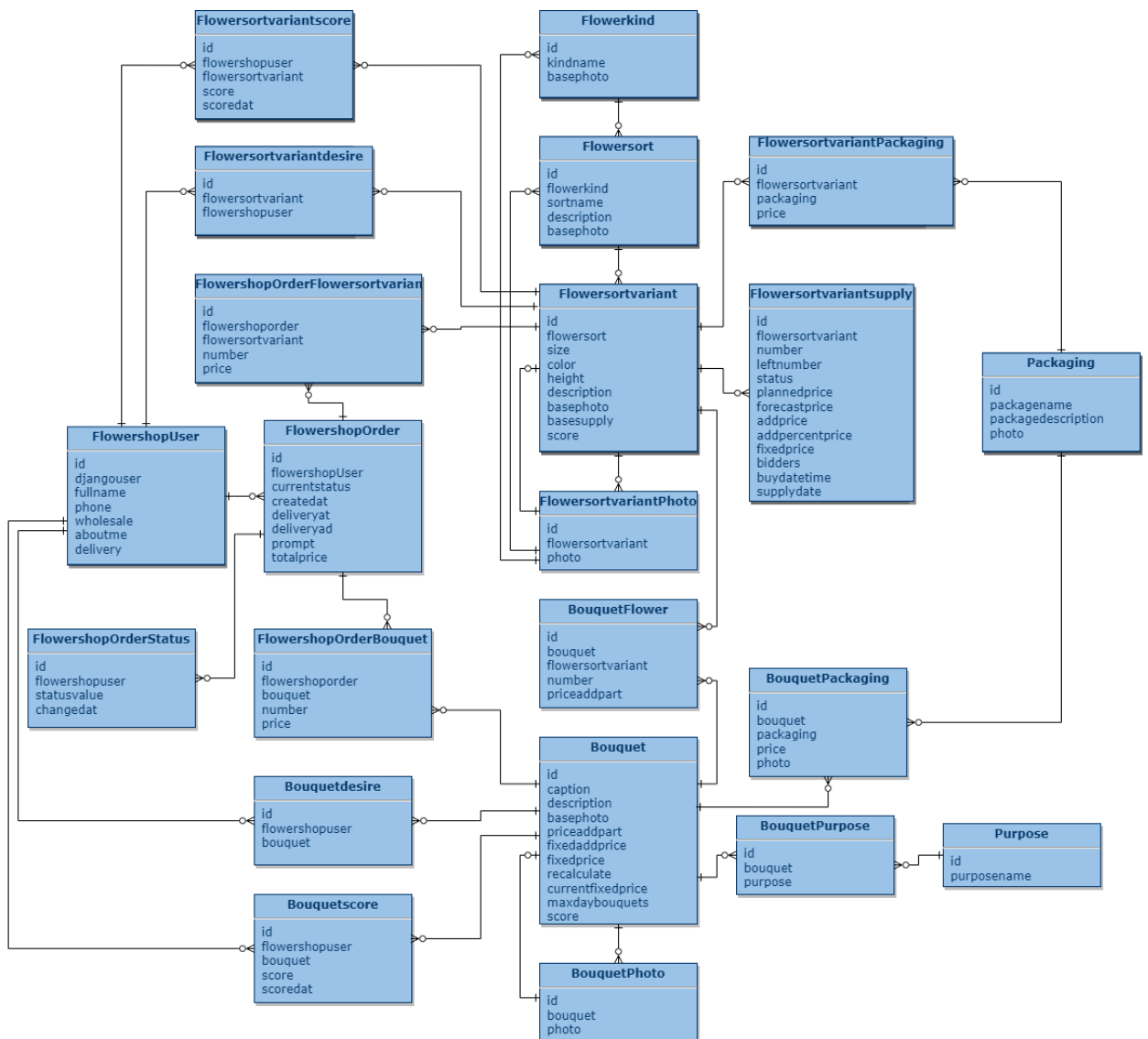


Рисунок 3.3 – Схема бази даних

3.4 Висновки за розділом 3

Реалізацію програми продажу живих квітів вирішено виконати через використання Django як фреймворку веброзробки та мови Python, порівнявши попередньо ці рішення разом з Wordpress та PHP. Моделювання сценаріїв роботи з програмою включало сценарії для представників магазину живих квітів та покупців. Структуру роботи з даними концептуально представлено як опис структури бази даних.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Порядок роботи з основними сутностями програми

Зважаючи на логіку, закладену у програму і пов'язану зі взаємодією між постачаннями та продажом, основними сутностями, які визначають послідовність реалізації функцій у програмі, є постачання та замовлення живих квітів.

Постачання живих квітів залежать від початкового плану, результатів аукціону, отримання та продажу живих квітів з постачання покупцям (рис. 4.1).

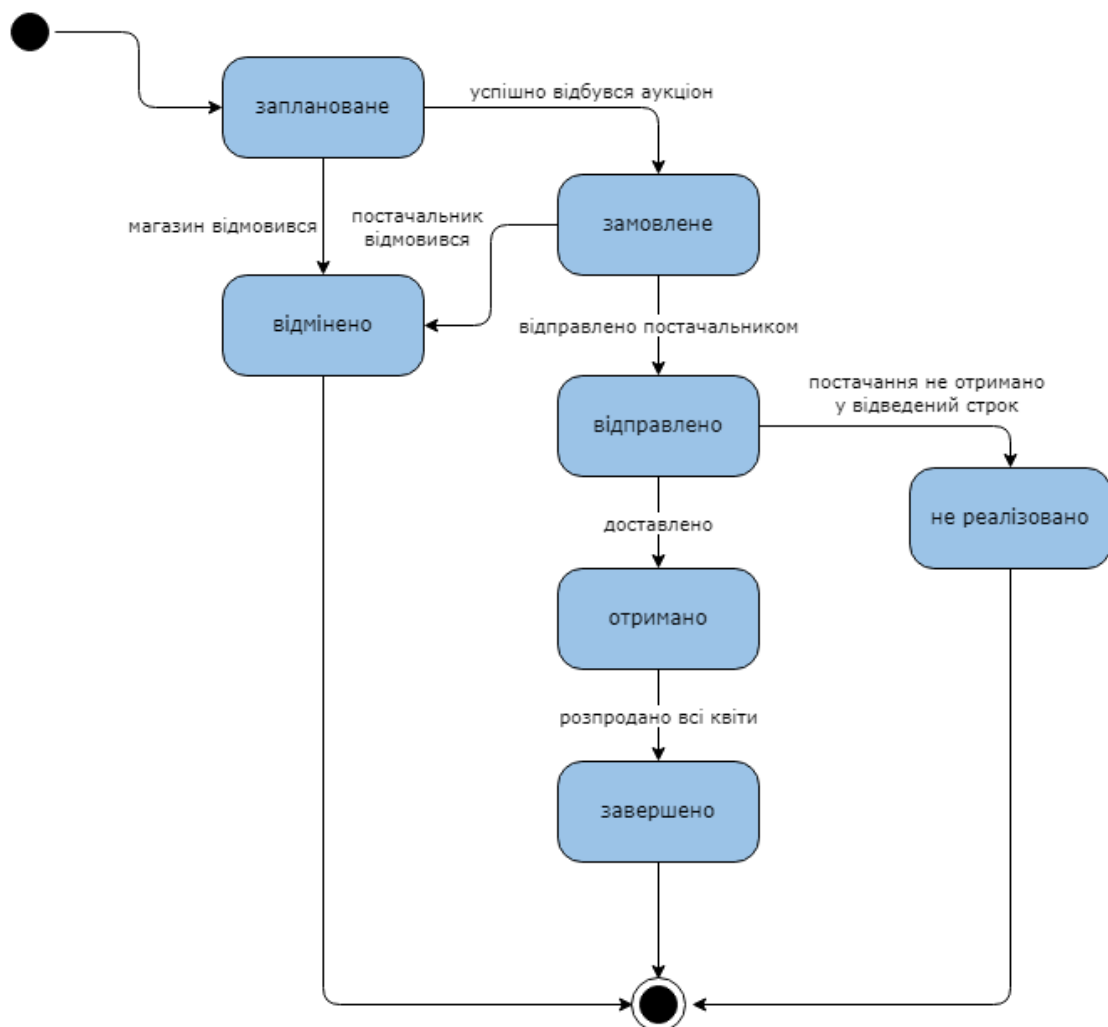


Рисунок 4.1 – Діаграма станів постачання живих квітів

Замовлення живих квітів виконуються покупцями, але потім мають бути опрацьовані представниками магазину, забрані з доставки або в магазині покупцем (рис. 4.2).

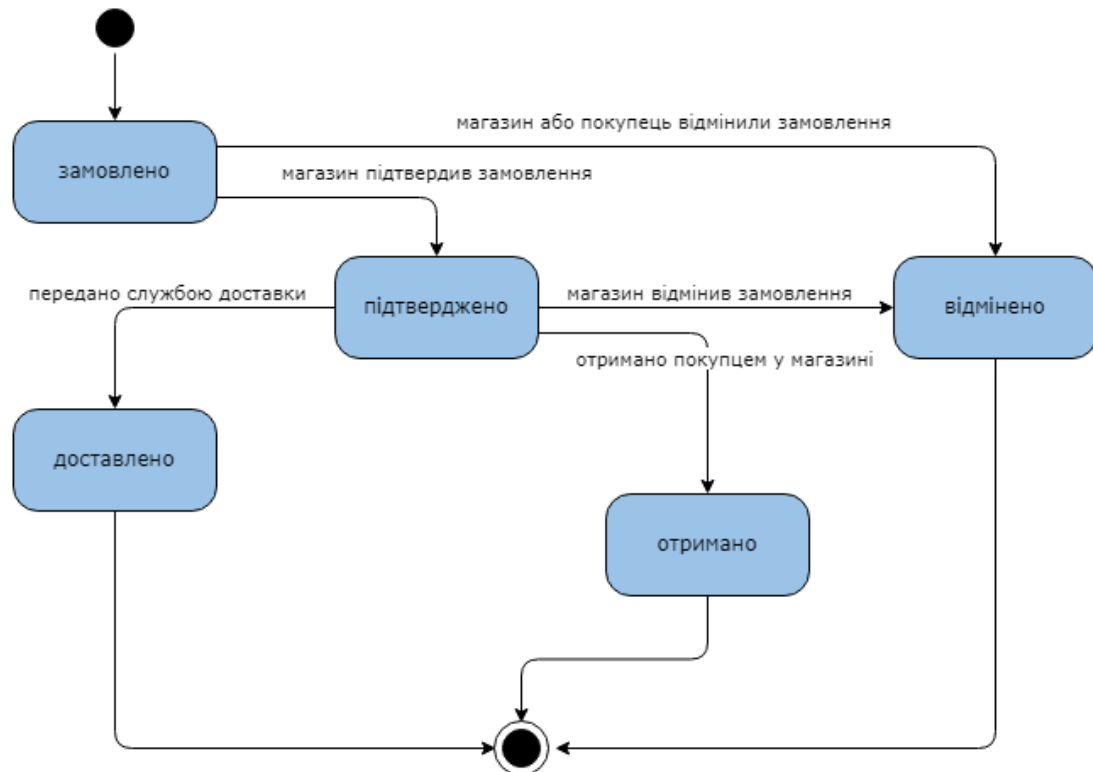


Рисунок 4.2 – Діаграма станів замовлення живих квітів

4.2 Рішення з реалізації вебзастосунку

Реалізація основної роботи з даними щодо забезпечення сценаріїв виконується в представленнях створеного Django-застосунку. Представлення використовують створені під ці сценарії шаблони вебсторінок для наповнення цих шаблонів підготовленими даними в представленнях.

Функція `flower_supply_add_view` забезпечує додавання нового постачання живих квітів на основі використання форми класу `FlowersortvariantsupplyForm`. У цій функції підтримується звернення до моделі, забезпеченої та навченої на основі запропонованого методу. Для цього виконується перетворення даних про участь в аукціоні, зазначених у формі, на запис з характеристиками вхідних даних моделі, викликається метод `predict` з

об'єкту моделі, куди передаються вхідні дані у вигляді одного запису в DataFrame. Отриманий результат обробляється та зберігається у таблицю бази даних.

Створення самої моделі виконується через засоби пакету scikit-learn, використовуючи його підпакет ensemble та клас RandomForestRegressor з нього. При створенні об'єкта моделі використовуються параметри n_estimators, що задає кількість дерев (обрану як 500), oob_score, що визначає необхідність оцінювання узагальнюючих властивостей моделі на основі коефіцієнта детермінації. Для виконання навчання викликається метод fit зі створеного об'єкта моделі, куди передаються значення вхідних ознак та значення вартості продажу цих живих квітів.

Функція choose_flower_view забезпечує відбір індивідуальних живих квітів. Через GET-запит в неї передаються такі параметри як перелік ідентифікаторів типів живих квітів (kinds), сортів живих квітів (sorts), потрібних розмірів живих квітів (sizes), кольорів живих квітів (colors), максимальної висоти квітів (maxheight), мінімальної висоти квітів (minheight), максимальна оцінка квітів (maxscore), мінімальна оцінка квітів (minscore), мінімальна вартість, яку передбачає сплатити користувач за квітку (minprice), максимальна вартість, яку передбачає сплатити користувач за квітку (maxprice), варіант сортування відібраних живих квітів (order), необхідність відображення тільки підсумкових сортів живих квітів чи повних варіантів (variant). Результати відбору, представлені або сортами живих квітів, або варіантами цих сортів, які є у продажі в магазині, разом з інформацією про задані сорти, переліком всіх наявних сортів живих квітів та типів цих квітів передаються в шаблон вебсторінки flowerchoose.html та передаються у відповідь. Шаблони вебсторінок реалізовані з підтримкою шаблону [11].

Функція flowershop_user_info забезпечує перегляд інформації про себе, якщо користувач не має прав представника магазину, та інформації про заданого користувача за його логіном через параметр userlogin (передається

через параметр GET-запиту), якщо користувач є представником магазину живих квітів.

Функція `choose_bouquet_view` забезпечує відбір потрібного букету живих квітів. Через GET-запит в неї передаються такі параметри як перелік ідентифікаторів цілей придбання букету (`purposes`), типів живих квітів (`kinds`), сортів живих квітів (`sorts`), потрібних розмірів живих квітів (`sizes`), кольорів живих квітів (`colors`), максимальної висоти квітів (`maxheight`), мінімальної висоти квітів (`minheight`), з яких складається букет, максимальна оцінка букету (`maxscore`), мінімальна оцінка букету (`minscore`), мінімальна вартість, яку передбачає сплатити користувач за букет (`minprice`), максимальна вартість, яку передбачає сплатити користувач за букет (`maxprice`), варіант сортування відібраних букетів живих квітів (`order`). Результати відбору, представлені букетами живих квітів, які є у продажі в магазині, разом з переліком всіх наявних сортів живих квітів та типів цих квітів передаються в шаблон вебсторінки `bouquetchoose.html` та передаються у відповідь.

Функція `flower_desire_add_view` забезпечує додавання букету живих квітів або сортів квітів до бажання, визначаючи параметр функції `flowerorbouquet`, що встановлює це букет чи сорт живих квітів, та `flower_id`, що встановлює ідентифікатор відповідно букета або сорту живих квітів. Якщо в базі даних відповідний сорт або букет живих квітів цим користувачем вже додано до бажання, то він вилючається з бажань.

Функція `flower_desires_view` забезпечує перегляд бажаних букетів або сортів живих квітів. Параметр GET-запиту `flowerorbouquet` встановлює це має бути відображено букети чи сорти живих квітів, а параметр `page` задає номер сторінки з результатами (результати по 10 на сторінці демонструються). Шаблон вебсторінки `desires.html` заповнюється результатами.

Функція `flower_scores_view` забезпечує перегляд самотійно оцінених букетів або сортів живих квітів, визначаючи через параметр GET-запиту `flowerorbouquet`, чи це мають бути сорти живих квітів у підсумку (0 для сортів живих квітів, 1 – для букетів, 2 – для одночасно всіх оцінок), чи букети, а через

параметри `minscore` та `maxscore` визначаючи інтервал оцінок, які мають відобразитися. Параметр `page` задає номер сторінки з результатами (результати по 10 на сторінці демонструються). Шаблон вебсторінки `scores.html` заповнюється результатами.

Функція `flowershop_user_orders_view` забезпечує перегляд своїх виконаних замовлень живих квітів, встановлюючи через параметр GET-запиту `status` потрібний стан замовлень (- 1 для всіх), а параметр `page` задає номер сторінки з результатами (результати по 10 на сторінці демонструються). Шаблон вебсторінки `myflowerorders.html` заповнюється результатами.

Функція `bouquet_view` забезпечує перегляд інформації про букет живих квітів з номером `bouquet_id`, який є параметром функції. Результати наповнюють шаблон вебсторінки `bouquet.html`, включаючи до них за ключем `bouquet` дані самого букету, `purposes` – перелік цілей, за якими рекомендовано купляти цей букет, `packagings` – доступні варіанти пакування для букету, `flowers` – склад букету (перелік варіантів сортів живих квітів), `photos` – фотографії букету, `scores` – виставлені оцінки користувачами букету живих квітів.

Функція `flower_view` забезпечує перегляд інформації про сорт живих квітів з номером `flower_id`, який є параметром функції. Результати наповнюють шаблон вебсторінки `flowersortvariant.html`, включаючи до них за ключем `flower` дані самого цього варіанта сорту живих квітів, `packagings` – доступні варіанти пакування для цього варіанта сорту живих квітів, `photos` – фотографії цього варіанта сорту живих квітів, `scores` – виставлені оцінки користувачами цьому варіанту сорту живих квітів, `supplies` – доступні постачання цього варіанта сорту живих квітів.

Функція `score_flower_view` забезпечує оцінювання букету або сорту живих квітів, визначаючи параметр функції `flowerorbouquet`, що встановлює це букет чи сорт живих квітів, `flower_id`, що встановлює ідентифікатор відповідно букета або сорту живих квітів, та `score`, що визначає саму оцінку.

Функція `flower_shop_order_set_status_view` забезпечує змінювання стану замовлення живих квітів для замовлення живих квітів з номером `orderid`, який є параметром функції, змінюючи його стан на `statusid`. При цьому відбувається контроль того, чи є цей користувач представником магазину живих квітів або чи він сам створив це замовлення, а також того, чи можливо за відповідного поточного стану змінити його на вказаний і саме цьому користувачу.

Функція `flower_shop_order_create_view` забезпечує замовлення живих квітів, при цьому не тільки зберігаючи саме замовлення, але також і спочатку контролюючи, що є доступна кількість потрібних квітів і букетів та вносячи зміни у базу даних щодо наявності квітів та букетів при збереженні замовлення, виконуючи за необхідності зв'язування варіантів сортів живих квітів, задіяних у замовленні, з наступним постачанням, з якого ці квіти мають бути взяті.

Функція забезпечує перегляд оцінених користувачем букетів або сортів живих квітів, визначаючи через параметр GET-запиту `flowerusername` логін облікового запису користувача, через параметр `flowerorbouquet`, чи це мають бути сорти живих квітів у підсумку (0 для сортів живих квітів, 1 – для букетів, 2 – для одночасно всіх оцінок), чи букети, а через параметри `minscore` та `maxscore` визначаючи інтервал оцінок, які мають відображатися, через параметри `start` та `end` – проміжок дат, коли відбулося оцінювання. Параметр `page` задає номер сторінки з результатами (результати по 10 на сторінці демонструються). Перевіряється, що користувач є представником магазину живих квітів. Шаблон вебсторінки `userscores.html` заповнюється результатами.

Функція `flowersortvariant_all_supplies_view` забезпечує перегляд запланованих та виконаних постачань живих квітів, перевіряючи наявність прав представника магазину живих квітів у користувача, та використовуючи параметр GET-запиту `executed` для визначення того, чи це заплановані, чи виконані повністю постачання, параметр `status` може визначати конкретний

стан постачань, які потрібно відобразити, `start` і `date` задають проміжок дат. Шаблон вебсторінки `flowersupplies.html` заповнюється результатами.

Функція `bouquet_add_view` забезпечує додавання інформації про букет живих квітів на основі використання класів `BouquetForm`, `BouquetPackagingForm`, `BouquetPurposeForm`, `BouquetPhotoForm`, `BouquetFlowerForm`, реалізуючи таким чином внесення даних про букет за всіма етапами, що включають базові дані, пакування, цілі придбання, фотографії та склад з живих квітів цього букету. Функція `bouquet_edit_view` забезпечує редагування інформації про букет живих квітів на основі аналогічної логіки, але вже для заданого номеру букету `bouquetid`.

Функція `flowershop_order_new_view` забезпечує замовлення живих квітів, при цьому реалізуючи окрім внесення самих даних замовлення перед цим перевірку того, що ці варіанти сортів живих квітів та букети доступні для продажу на потрібний час та перевіряючи, що для цього користувача доступне придбання квітів у такій кількості (програмно задано кількість і якщо в замовленні вона більше, то покупець повинен мати позначку, що він є гуртовим покупцем). Після внесення замовлення відповідна кількість квітів відкидається з доступної кількості в базі даних (зокрема з постачання відповідного варіанта, а якщо в постачанні залишається 0 одиниць, то визначається наступне базове постачання для цього варіанта сорту живих квітів).

Функція `flowersort_add_view` забезпечує додавання сортів живих квітів на основі роботи з формою `FlowersortForm`. Функція `flowersort_edit_view` забезпечує редагування інформації про сорт живих квітів з номером `flowersortid`.

Функція `flower_shop_order_search_view` забезпечує пошук замовлення живих квітів, де в GET-запиті в параметрі `orderid` задається номер замовлення, якщо він відомий, у параметрах `start` і `end` інтервал часу, коли могло бути створено замовлення, `startdeliv` і `enddeliv` – інтервал часу, коли мало бути доставлено замовлення, `minprice` і `maxprice` – інтервал вартості замовлення,

username – логін користувача, що створив замовлення живих квітів, userphone – номер телефону користувача. Результати надаються в шаблон вебсторінки searchflowerorder.html тільки тоді, коли користувач є представником магазину живих квітів.

Функція flower_shop_order_detail_view забезпечує перегляд інформації про замовлення живих квітів з номером orderid, який є параметром функції, при цьому дані передаються в шаблон вебсторінки flowerorder.html тільки тоді, коли користувач є представником магазину живих квітів або коли він сам створив це замовлення.

Функція flowershop_orders_view забезпечує перегляд замовлень живих квітів у магазині, встановлюючи через параметр GET-запиту status потрібний стан замовлень (- 1 для всіх), параметр page задає номер сторінки з результатами (результати по 10 на сторінці демонструються), параметр start задає дату і час, з якого мало відбуватися створення замовлення, а end – відповідно кінець цього інтервалу, параметр startdeliv задає дату і час, з якого мало відбуватися доставлення замовлення, а enddeliv – відповідно кінець цього інтервалу. Шаблон вебсторінки flowerorders.html заповнюється результатами.

Функція search_flower_view забезпечує пошук серед сортів живих квітів для представників магазину живих квітів. Тому в ній спочатку контролюється, що користувач є представником магазину живих квітів. Якщо це так, то відбувається формування результатів пошуку. Для цього через GET-запит в неї передаються такі параметри як ідентифікаційний номер варіанта сорту живих квітів, якщо він відомий (variantid), перелік ідентифікаторів типів живих квітів (kinds), сортів живих квітів (sorts), потрібних розмірів живих квітів (sizes), кольорів живих квітів (colors), максимальної висоти квітів (maxheight), мінімальної висоти квітів (minheight), максимальна оцінка квітів (maxscore), мінімальна оцінка квітів (minscore), мінімальна вартість, яку передбачає сплатити користувач за квітку (minprice), максимальна вартість, яку передбачає сплатити користувач за квітку (maxprice), варіант сортування відібраних живих квітів (order), необхідність відображення тільки

підсумкових сортів живих квітів чи повних варіантів (variant). Результати відбору, представлені або сортами живих квітів, або варіантами цих сортів разом з інформацією про задані сорти, переліком всіх наявних сортів живих квітів та типів цих квітів передаються в шаблон вебсторінки `flowersearch.html` та передаються у відповідь.

Функція `search_bouquet_view` забезпечує пошук серед букетів живих квітів для представників магазину живих квітів. Тому в ній спочатку контролюється, що користувач є представником магазину живих квітів. Якщо це так, то відбувається формування результатів пошуку. Для цього через GET-запит в неї передаються такі параметри як назва букету або частина цієї назви (name), ідентифікаційний номер букету, якщо він відомий (id), перелік ідентифікаторів типів живих квітів (kinds), сортів живих квітів (sorts), потрібних розмірів живих квітів (sizes), кольорів живих квітів (colors), максимальної висоти квітів (maxheight), мінімальної висоти квітів (minheight), з яких складається букет, максимальна оцінка букету (maxscore), мінімальна оцінка букету (minscore), мінімальна вартість, яку передбачає сплатити користувач за букет (minprice), максимальна вартість, яку передбачає сплатити користувач за букет (maxprice), варіант сортування відібраних букетів живих квітів (order). Результати відбору, представлені букетами живих квітів разом з переліком всіх наявних сортів живих квітів та типів цих квітів передаються в шаблон вебсторінки `bouquetsearch.html` та передаються у відповідь.

Розроблено для програми продажу живих квітів наступні моделі для роботи з даними.

Модель виду живих квітів реалізована класом `Flowerkind`.

Поле класу `kindname` є об'єктом класу `CharField` та задає назву виду живих квітів.

Поле класу `basephoto` є об'єктом класу `ForeignKey` та задає фотографію варіації сорту живих квітів, яка використовується для ілюстрації цього виду живих квітів при відображенні такої фотографії на сторінці, через з'єднання з моделлю класу `FlowersortvariantPhoto`.

Модель сортів живих квітів реалізована класом `Flowersort`.

Поле `flowerkind` є об'єктом класу `ForeignKey` та задає вид живих квітів, до якого належить цей сорт, через з'єднання з моделлю класу `Flowerkind`.

Поле `sortname` є об'єктом класу `CharField` та задає назву сорту живих квітів.

Поле `description` є об'єктом класу `CharField` та задає опис особливостей цього сорту живих квітів.

Поле `basephoto` є об'єктом класу `ForeignKey` та задає фотографію варіації сорту живих квітів, яка використовується для ілюстрації цього сорту живих квітів при відображенні такої фотографії на сторінці, через з'єднання з моделлю класу `FlowersortvariantPhoto`.

Модель варіації сорту живих квітів, яка є позицією в каталогі магазину, реалізована класом `Flowersortvariant`.

Поле `flowersort` є об'єктом класу `ForeignKey` та задає сорт квітів, варіацією якого є дана, через з'єднання з моделлю класу `Flowersort`.

Поле `size` є об'єктом класу `PositiveSmallIntegerField` та задає розмір квітки у цієї варіації даного сорту живих квітів.

Поле `color` є об'єктом класу `PositiveSmallIntegerField` та задає колір квітки у цієї варіації даного сорту живих квітів.

Поле `height` є об'єктом класу `PositiveSmallIntegerField` та задає висоту (довжина ніжки) цієї варіації даного сорту живих квітів.

Поле `description` є об'єктом класу `TextField` та задає опис особливостей цієї варіації.

Поле `basephoto` є об'єктом класу `ForeignKey` та задає фотографію цієї варіації сорту живих квітів, яка використовується при відображенні такої фотографії на сторінці, через з'єднання з моделлю класу `FlowersortvariantPhoto`.

Поле `basesupply` є об'єктом класу `ForeignKey` та задає постачання цієї варіації цього сорту живих квітів, з якого в замовленнях використовуються

квіти на даний момент, через з'єднання з моделлю класу `Flowersortvariantsupply`.

Поле `score` є об'єктом класу `DecimalField` та задає середню оцінку, отриману від користувачів.

Модель варіацій сортів живих квітів реалізована класом `FlowersortvariantPhoto`.

Поле `flowersortvariant` є об'єктом класу `ForeignKey` та задає варіацію сорту живих квітів, яка показана на фотографії, через з'єднання з моделлю класу `Flowersortvariant`.

Поле `photo` є об'єктом класу `ImageField` та задає дані фотографії (саме зображення).

Модель пакувань, які доступні для конкретної варіації сорту живих квітів, реалізована класом `FlowersortvariantPackaging`.

Поле `flowersortvariant` є об'єктом класу `ForeignKey` та задає варіацію сорту живих квітів, для якої доступне це пакування, через з'єднання з моделлю класу `Flowersortvariant`.

Поле `packaging` є об'єктом класу `ForeignKey` та задає пакування, в яке можна оформити цю варіацію живих квітів, через з'єднання з моделлю класу `Packaging`.

Поле `price` є об'єктом класу `DecimalField` та задає вартість самого пакування цієї варіації сорту живих квітів.

Модель постачання варіацій сортів живих квітів реалізована класом `Flowersortvariantsupply`.

Поле `flowersortvariant` є об'єктом класу `ForeignKey` та задає варіацію сорту живих квітів, постачання якої здійснюється в даному випадку, через з'єднання з моделлю класу `Flowersortvariant`.

Поле `number` є об'єктом класу `PositiveIntegerField` та задає кількість одиниць живих квітів у цій варіації даного сорту, що закуплена і має бути поставлена для продажу.

Поле `leftnumber` є об'єктом класу `PositiveIntegerField` та задає кількість одиниць живих квітів у цій варіації даного сорту, яка ще на даний момент не є проданою або замовленою.

Поле `status` є об'єктом класу `PositiveSmallIntegerField` та задає стан цього постачання цієї варіації живих квітів.

Поле `plannedprice` є об'єктом класу `DecimalField` та задає заплановану вартість придбання одиниці цієї варіації живих квітів.

Поле `forecastprice` є об'єктом класу `DecimalField` та задає прогнозовану вартість придбання одиниці цієї варіації живих квітів.

Поле `fixedprice` є об'єктом класу `DecimalField` та задає фактичну вартість, за якою буде продаватися одиниця цієї варіації живих квітів.

Поле `addpercentprice` є об'єктом класу `DecimalField` та задає відсоток націнки на вартість придбання цієї варіації живих квітів у постачальників для формування вартості подальшого продажу магазином.

Поле `addprice` є об'єктом класу `DecimalField` та задає додаткову націнку (абсолютне значення) на вартість придбання цієї варіації живих квітів у постачальників для формування вартості подальшого продажу магазином.

Поле `bidders` є об'єктом класу `PositiveIntegerField` та задає кількість учасників аукціону з продажу цієї варіації живих квітів.

Поле `buydatetime` є об'єктом класу `DateTimeField` та задає дату і час проведення аукціону живих квітів.

Поле `supplydate` є об'єктом класу `DateField` та задає дату, коли має бути поставлено в магазин від постачальника цю партію живих квітів.

Модель цілі, за якою можуть куплятися букети живих квітів, реалізована класом `Purpose` з полем `purposename`, що є об'єктом класу `CharField` та задає назву (формулювання) цілі придбання живих квітів.

Модель букетів живих квітів у магазині реалізована класом `Bouquet`.

Поле `caption` є об'єктом класу `CharField` та задає назву букету живих квітів.

Поле `description` є об'єктом класу `TextField` та задає опис букету живих квітів.

Поле `basephoto` є об'єктом класу `ForeignKey` та задає фотографію цього букету живих квітів, яка використовується при відображенні такої фотографії на сторінці, через з'єднання з моделлю класу `BouquetPhoto`.

Поле `priceaddpart` є об'єктом класу `DecimalField` та задає відсоток націнки на сумарну вартість окремих квітів при формуванні вартості букету.

Поле `fixedaddprice` є об'єктом класу `DecimalField` та задає додаткову націнку (абсолютне значення) на вартість окремих квітів при формуванні вартості букету.

Поле `fixedprice` є об'єктом класу `DecimalField` та задає підсумкову стандартну вартість продажу букету.

Поле `recalculate` є об'єктом класу `BooleanField` та задає позначку про те, чи треба переобчислювати вартість букету при постачанні нових живих квітів, чи потрібну використовувати значення з поля `fixedprice` постійно.

Поле `currentfixedprice` є об'єктом класу `DecimalField` та задає підсумкову акційну вартість продажу букету.

Поле `maxdaybouquets` є об'єктом класу `PositiveIntegerField` та задає та задає максимальна кількість букетів, яка може бути продана на день.

Поле `score` є об'єктом класу `DecimalField` та задає середню оцінку, отриману від користувачів.

Модель цілей, за якими рекомендовано купувати конкретні букети живих квітів, реалізована класом `BouquetPurpose`.

Поле `bouquet` є об'єктом класу `ForeignKey` та задає букет живих квітів, для купівлі якого рекомендована ця ціль, через з'єднання з моделлю класу `Bouquet`.

Поле `purpose` є об'єктом класу `ForeignKey` та задає ціль, за якою рекомендовано купляти цей букет живих квітів, через з'єднання з моделлю класу `Purpose`.

Модель можливих конкретних пакувань конкретних букетів живих квітів реалізована класом `BouquetPackaging`.

Поле `bouquet` є об'єктом класу `ForeignKey` та задає букет живих квітів, пакування для якого визначено, через з'єднання з моделлю класу `Bouquet`.

Поле `packaging` є об'єктом класу `ForeignKey` та задає пакування, яке може використовуватися для цього букету живих квітів, через з'єднання з моделлю класу `Packaging`.

Поле `price` є об'єктом класу `DecimalField` та задає вартість такого пакування.

Поле `photo` є об'єктом класу `ImageField` та задає дані фотографії (саме зображення) цього пакування букету живих квітів.

Модель живих квітів, які входять до букету, реалізована класом `BouquetFlower`.

Поле `bouquet` є об'єктом класу `ForeignKey` та задає букет живих квітів, до якого входить ця варіація сорту живих квітів, через з'єднання з моделлю класу `Bouquet`.

Поле `flowersortvariant` є об'єктом класу `ForeignKey` та задає варіацію сорту живих квітів, що входить до складу цього букету, через з'єднання з моделлю класу `Flowersortvariant`.

Поле `number` є об'єктом класу `PositiveIntegerField` та задає кількість одиниць цієї варіації живих квітів, що використовується в даному букеті.

Поле `priceaddpart` є об'єктом класу `DecimalField` та задає відсоток націнки на одиницю цієї квітки при формуванні вартості букету.

Модель фотографії букетів живих квітів реалізована класом `BouquetPhoto`.

Поле `bouquet` є об'єктом класу `ForeignKey` та задає букет живих квітів, який зображений на фотографії, через з'єднання з моделлю класу `Bouquet`.

Поле `photo` є об'єктом класу `ImageField` та задає дані фотографії (саме зображення).

Модель користувачів програми продажу живих квітів реалізована класом `FlowershopUser` на основі наслідування Django-класу `User` з підпаketу `django.contrib.auth.models`.

Поле `django_user` є об'єктом класу `OneToOneField` та задає обліковий запис Django, за яким авторизується в програмі цей користувач через з'єднання з моделлю класу `User`.

Поле `fullname` є об'єктом класу `CharField` та задає повне ім'я користувача програми продажу живих квітів.

Поле `phone` є об'єктом класу `CharField` та задає номер телефону користувача програми продажу живих квітів.

Поле `wholesale` є об'єктом класу `BooleanField` та задає позначку про те, що цей користувач представляє гуртового покупця живих квітів.

Поле `aboutme` є об'єктом класу `TextField` та задає опис користувачем програми продажу живих квітів себе.

Поле `delivery` є об'єктом класу `CharField` та задає адресу доставки, яка може використовуватися під час формування замовлення живих квітів стандартно.

Поле `flowersortvariants` є об'єктом класу `ManyToManyField` та задає варіації сортів живих квітів, які були збережені користувачем у бажання, через з'єднання з моделлю класу `Flowersortvariant`.

Поле `bouquets` є об'єктом класу `ManyToManyField` та задає букети живих квітів, які були збережені у бажання цим користувачем, через з'єднання з моделлю класу `Bouquet`.

Модель замовлення живих квітів реалізована класом `FlowershopOrder`.

Поле `flowershop_user` є об'єктом класу `ForeignKey` та задає користувача програми продажу живих квітів, який сформував замовлення, через з'єднання з моделлю класу `FlowershopUser`.

Поле `currentstatus` є об'єктом класу `PositiveSmallIntegerField` та задає поточний стан роботи над цим замовленням живих квітів.

Поле `createdat` є об'єктом класу `DateField` та задає дату і час, коли було створено це замовлення живих квітів.

Поле `deliveryat` є об'єктом класу `DateField` та задає дату і час, коли має бути готове це замовлення живих квітів.

Поле `deliveryad` є об'єктом класу `CharField` та задає адресу доставки замовлення букету живих квітів, якщо наявна служба доставки;

Поле `prompt` є об'єктом класу `CharField` та задає підказку щодо особливостей цього замовлення живих квітів.

Поле `totalprice` є об'єктом класу `DecimalField` та задає вартість цього замовлення живих квітів.

Модель зміни станів замовлень живих квітів реалізована класом `FlowershopOrderStatus`.

Поле `flowershopuser` є об'єктом класу `ForeignKey` та задає користувача, який вніс цю зміну стану замовлення живих квітів, через з'єднання з моделлю класу `FlowershopUser`.

Поле `statusvalue` є об'єктом класу `PositiveSmallIntegerField` та задає значення стану, яке було встановлено.

Поле `changedat` є об'єктом класу `DateField` та задає дату і час, коли ця зміна стану замовлення живих квітів була виконана.

Модель букетів живих квітів, що входять до складу замовлення живих квітів, реалізована класом `FlowershopOrderBouquet`.

Поле `flowershoporder` є об'єктом класу `ForeignKey` та задає замовлення живих квітів, в яке входить цей букет, через з'єднання з моделлю класу `FlowershopOrder`.

Поле `bouquet` є об'єктом класу `ForeignKey` та задає букет живих квітів, який входить в це замовлення, через з'єднання з моделлю класу `FlowershopUser`.

Поле `number` є об'єктом класу `PositiveIntegerField` та задає кількість таких букетів живих квітів у замовленні.

Поле `price` є об'єктом класу `DecimalField` та задає вартість одного букету живих квітів.

Модель варіації сортів живих квітів, що входять до складу замовлення живих квітів, реалізована класом `FlowershopOrderFlowersortvariant`.

Поле `flowershoporder` є об'єктом класу `ForeignKey` та задає замовлення живих квітів, в яке входить ця варіація сорту, через з'єднання з моделлю класу `FlowershopOrder`.

Поле `flowersortvariant` є об'єктом класу `ForeignKey` та задає варіацію сорту живих квітів, яка входить до замовлення, через з'єднання з моделлю класу `Flowersortvariant`.

Поле `number` є об'єктом класу `PositiveIntegerField` та задає кількість одиниць квітів цієї варіації даного сорту в замовленні.

Поле `price` є об'єктом класу `DecimalField` та задає вартість одиниці квітів цієї варіації даного сорту в замовленні.

Модель оцінювання варіацій сортів живих квітів користувачами програми продажу живих квітів реалізована класом `Flowersortvariantscore`.

Поле `flowershopuser` є об'єктом класу `ForeignKey` та задає користувача програми продажу живих квітів, який оцінив цю варіацію сорту живих квітів, через з'єднання з моделлю класу `FlowershopUser`.

Поле `flowersortvariant` є об'єктом класу `ForeignKey` та задає варіацію сорту живих квітів, яка була оцінена користувачем програми продажу живих квітів, через з'єднання з моделлю класу `Flowersortvariant`.

Поле `score` є об'єктом класу `PositiveSmallIntegerField` та задає виставлену оцінку.

Поле `scoredate` є об'єктом класу `DateTimeField` та задає дату, час, коли оцінка варіації живих квітів була виставлена користувачем програми продажу живих квітів.

Модель оцінювання букетів живих квітів користувачами програми продажу живих квітів реалізована класом `Bouquetscore`.

Поле `flowershopuser` є об'єктом класу `ForeignKey` та задає користувача програми продажу живих квітів, який оцінив цей букет живих квітів, через з'єднання з моделлю класу `FlowershopUser`.

Поле `bouquet` є об'єктом класу `ForeignKey` та задає букет живих квітів, який був оцінений користувачем програми продажу живих квітів, через з'єднання з моделлю класу `Bouquet`.

Поле `score` є об'єктом класу `PositiveSmallIntegerField` та задає виставлена оцінка.

Поле `scoredate` є об'єктом класу `DateTimeField` та задає дату, час, коли оцінка букету живих квітів була виставлена користувачем програми продажу живих квітів.

Модель доступних пакувань живих квітів реалізована класом `Packaging`.

Поле `packagename` є об'єктом класу `CharField` та задає назву пакування живих квітів.

Поле `packagedescription` є об'єктом класу `CharField` та задає додатковий опис особливостей цього пакування.

Поле `photo` є об'єктом класу `ImageField` та задає дані загальної фотографії цього пакування, представляючи таким чином саме зображення у вигляді даних.

4.3 Висновки за розділом 4

Реалізовано програму продажу живих квітів, яка забезпечує як реалізацію основних користувацьких функцій щодо придбання та продажу квітів, так і прогнозування вартості постачання живих квітів на основі виконаної реалізації запропонованого методу. Описано логіку створених програмних функцій.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Опис програми

Програмне забезпечення продажу живих квітів надає можливість відбирати потрібні букети або сорти живих квітів, замовляти такі букети або квіти індивідуально, керувати цими замовленнями зокрема зі сторони магазину, визначати наявність та цілісні постачання живих квітів від зовнішніх постачальників через придбання квітів на аукціоні. Робота підтримується як для індивідуальних покупців, так і для великих гуртових покупців.

5.2 Експлуатація програми

Для експлуатації програми необхідно звернутися до адреси, за якою розміщено вебсайт. Керування взаємодією в подальшому реалізується через меню, доступ до якого відкривається при натисканні на іконку з 3 горизонтальними чорними смужками в правому верхньому куті (рис. 5.1). Меню включає пункти «Букети», що дозволяє перейти до відбору букету живих квітів, визначаючи різні критерії відбору (сорти, типи квітів, кольори, розміри, вартість тощо), «Квіти», що дозволяє аналогічний відбір організувати для варіантів сортів живих квітів, «Замовлення», звідки реалізується доступ до переліку замовлень цього користувача, а також до формування нового замовлення на основі тих квітів та букетів, які раніше були додані в замовлення зі сторінок цих букетів та квітів, «Особистий кабінет», де відображається інформація про користувача, яку можна відредагувати за потреби, а також наявні кнопки для відображення бажань та виконаних оцінок користувачем, «Вихід» для виходу з облікового запису. Якщо користувач не авторизувався, то демонструється надпис «Вхід».

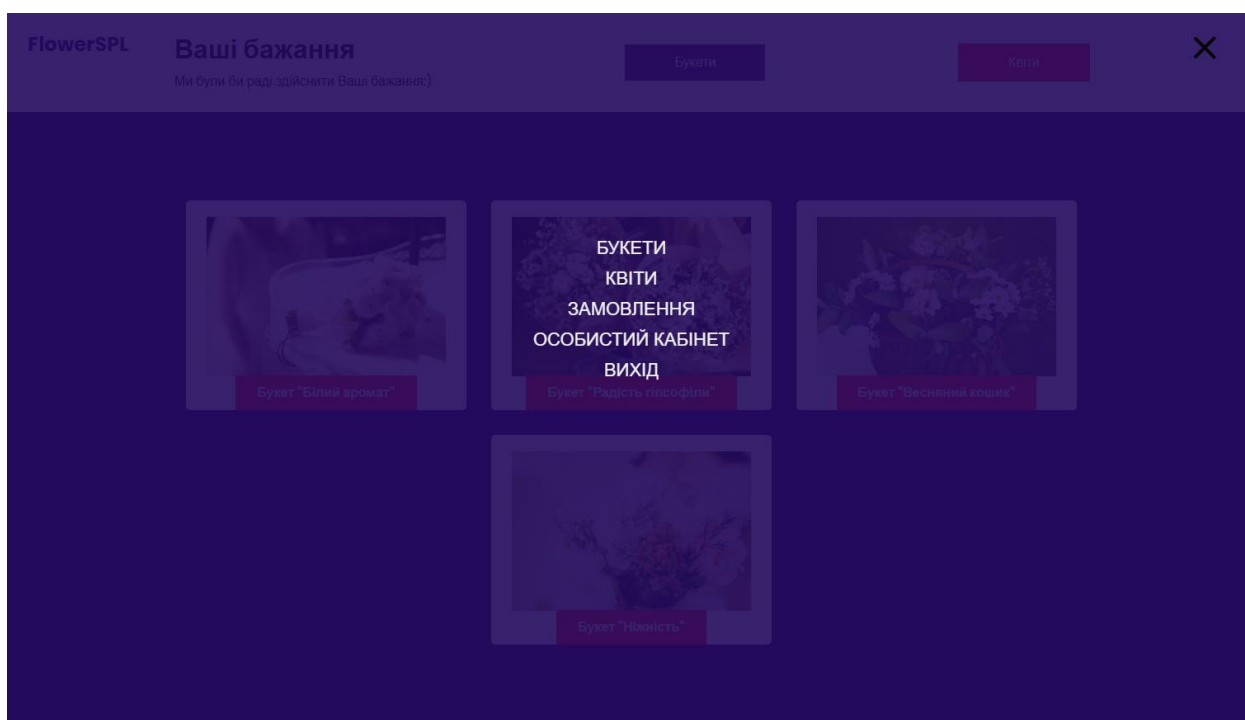


Рисунок 5.1 – Меню

Це варіант меню для покупця. Для представника магазину живих квітів пункти з квітами та букетами дозволяють виконувати пошук, а не відбір. У пошуку можна задавати відповідні назви та ідентифікаційні номери відповідно букету або варіанту сорту живих квітів. Окрім того з цих сторінок можна переходити до додавання нових букетів, варіантів живих квітів, а також редагувати існуючі. Міститься також додатковий пункт «Постачання», що дозволяє визначати нові постачання живих квітів, отримувати таким чином доступ до прогнозування вартості постачання квітів на основі проведеного аукціону, редагувати існуючі постачання.

При виборі сторінки з бажаннями з особистого кабінету покупець може обирати або букети, або квіти, таким чином визначаючи, які саме бажання необхідно відобразити (рис. 5.2). Додавання до бажань відбувається зі сторінки букету або варіанту сорту живих квітів, а видалення можна виконати, натискаючи на назву букету. Відкрити ж сторінку букету можна, натиснувши на фотографію букету.

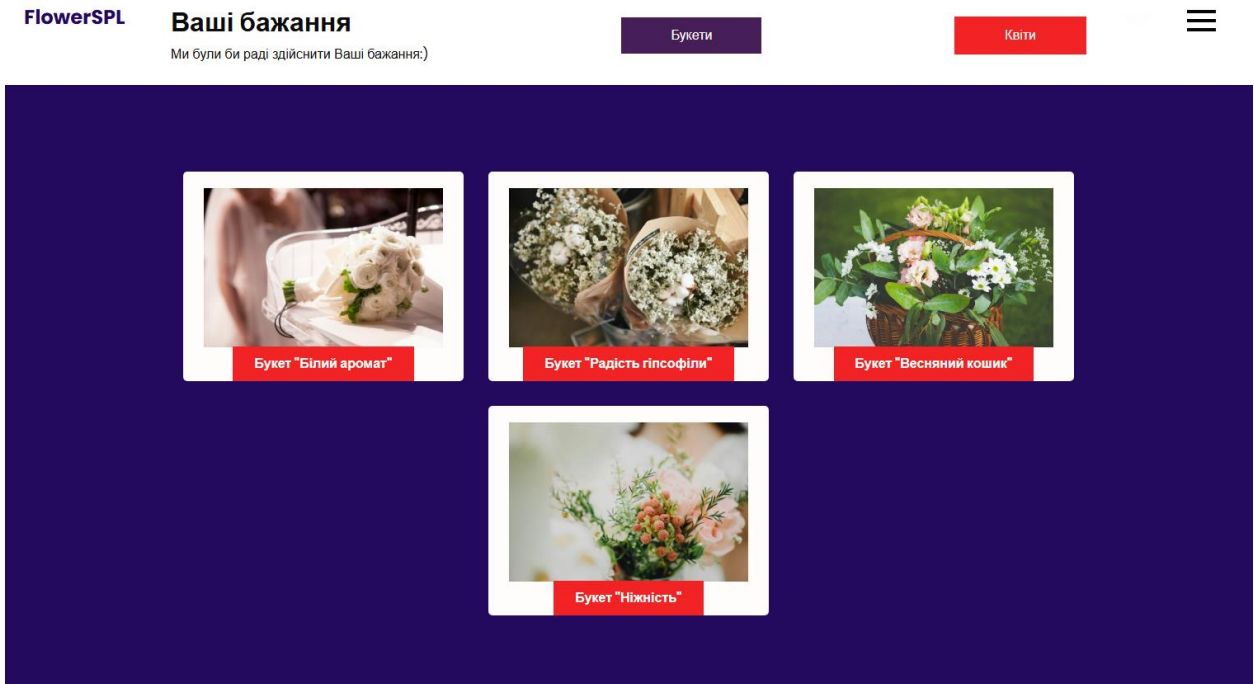


Рисунок 5.2 – Відтворення власних бажань авторизованого користувача

На сторінці букету живих квітів (рис. 5.3) відображається його назва, опис букету, фотографії (якщо доступно декілька фотографій, то вони відображаються в каруселі, тобто з'являються з боків стрілки для перемикання між фотографіями), кнопка для додавання або вилучення з бажань букету, вибір варіанту фасування букету, що дозволяє відобразити вартість такого фасування, вартість всього букету, а також вартість букету в майбутньому. Вона позначена як наступна вартість букету. Наступна вартість букету обчислюється за результатами квітів у наступних постачаннях після поточного. Відповідно для розрахунку береться або вже нова зафіксована вартість, або вартість за результатами прогнозування (або запланована встановлена, якщо представник магазину вирішив встановити власну вартість, змінюючи значення прогнозу). Таким чином, для покупця саме в цьому полі відображається результат використання моделі прогнозування вартості постачання живих квітів, сформованої на основі запропонованого методу. На сторінці також відображається оцінка та кнопка для переходу до всіх виставлених оцінок, що демонструються внизу сторінки, кнопка для додавання букету до поточного замовлення (потім потрібно перейти в розділ

замовлень та вибрати сторінку нового замовлення), склад букету, перелік цілей, за якими цей букет рекомендовано купляти.

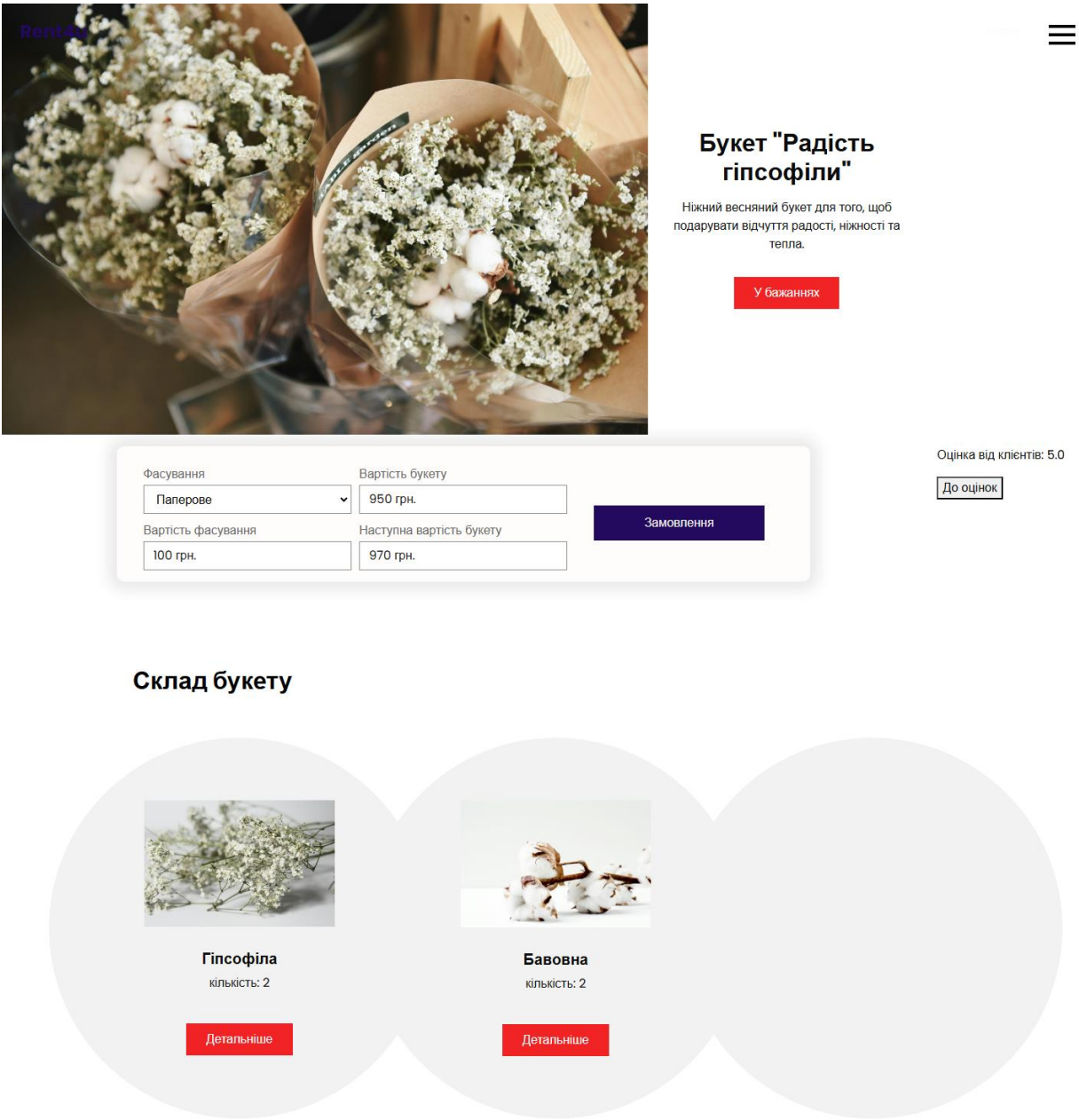


Рисунок 5.3 – Сторінка букету живих квітів

Для представника магазину живих квітів доступна сторінка для керування замовленнями живих квітів (рис. 5.4), що має розширені засоби переключення між варіантами роботи з замовленнями, включаючи пошук замовлень, коли можна задати різні параметри. Сам же перелік замовлень

дозволяє перейти до конкретного замовлення, звідки також можна перемикатись між описаними вище функціями.

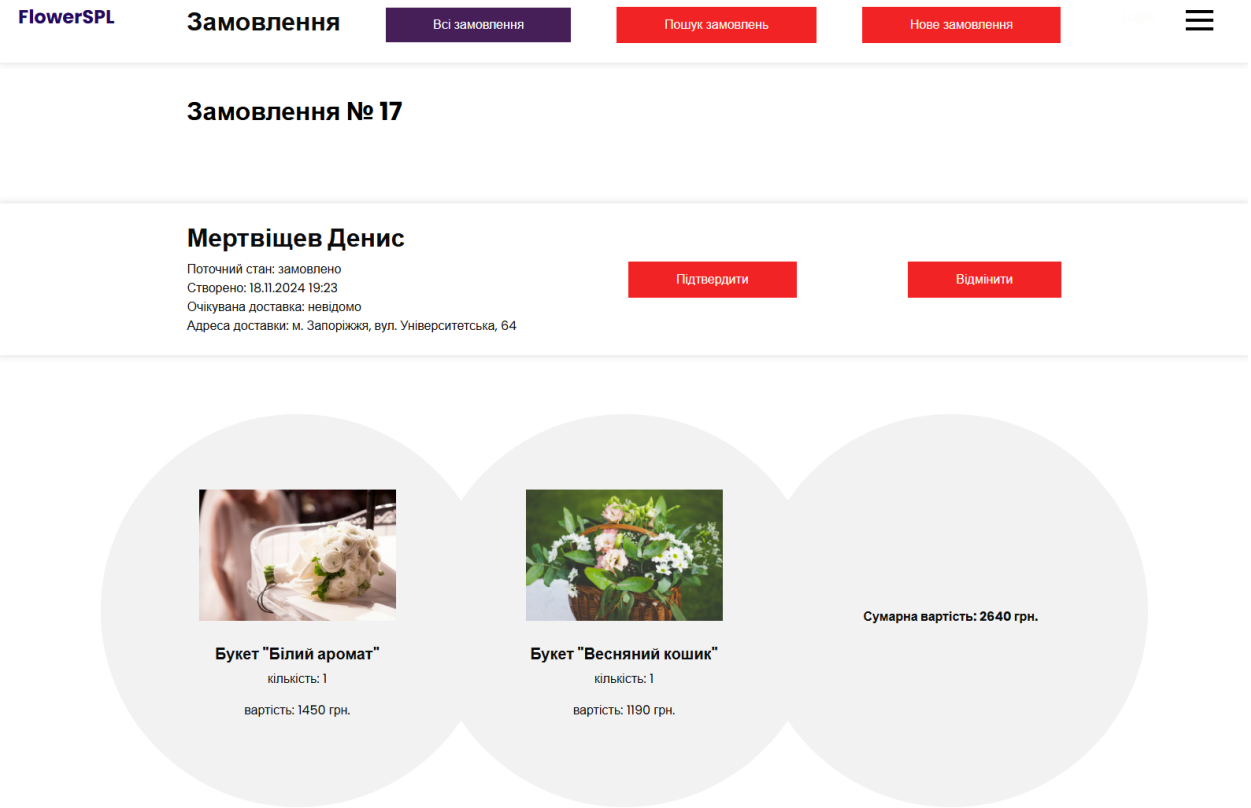


Рисунок 5.4 – Сторінка замовлень у магазині живих квітів

Зі сторінки замовлення доступний його ідентифікаційний номер, інформація про замовника, при чому можна натиснути на ім'я та відкрити сторінку з даними про нього, оціненими букетами та квітами. Також надаються дані поточного стану, дата і час створення, дата доставки, якщо така дата встановлена. А якщо стан замовлення є підтвердженим, то надається кнопка для появи поля введення значення дати. Також надається адреса доставки, склад замовлення. Надаються кнопки для зміни стану замовлення. Набір кнопок залежить від поточного стану. Також внизу сторінки доступне посилання для відображення історії зміни станів цього замовлення живих квітів.

5.3 Експериментальне дослідження

Для експериментального дослідження було використано дані з вибірки [4], де представлені результати торгів на аукціоні з продажу квітів у Нідерландах, які були в подальшому розширені. Результати аукціону стосуються продажу двох типів бегоній. Кожен аукціон передбачає гуртовий продаж квітів, які надані різними постачальниками. Квіти продаються послідовно, фактично випадково представляючи позиції кожного постачальника. Аукціон продовжується до того моменту, поки всі квіти будуть продані, а починається о 6:30, коли вже всі квіти від постачальників мають бути готові для продажу. Відповідно квіти кожного постачальника виставляються на продаж, визначається шляхом аукціону покупець та ціна продажу, а тоді відбувається перехід до наступної партії.

Дані представлені записами, кожен з яких містить наступні ознаки:

- вартість продажу одиниці;
- кількість учасників аукціону;
- час початку аукціону (в секундах, що пройшли з 6:30, коли розпочались всі аукціони відповідного дня);
- відповідність першому типу бегоній: 1, якщо це бегонії цього типу, та 0 у протилежному випадку;
- відповідність другому типу бегоній: 1, якщо це бегонії цього типу, та 0 у протилежному випадку;
- продаж у понеділок: 1, якщо аукціон відбувався у понеділок, та 0 у протилежному випадку;
- продаж у вівторок: 1, якщо аукціон відбувався у вівторок, та 0 у протилежному випадку;
- продаж у середу: 1, якщо аукціон відбувався у середу, та 0 у протилежному випадку;
- продаж у четвер: 1, якщо аукціон відбувався у четвер, та 0 у протилежному випадку.

Для експериментального дослідження було виділено ряд моделей, які на основі вхідних ознак, повинні були спрогнозувати вартість продажу одиниці з партії квітів на кожному аукціоні, включаючи:

- моделі на основі дерев рішень;
- метод опорних векторів у варіанті для розв’язання задачі регресії;
- запропонований метод на основі створення моделей за методом Random Forest.

Кожен аукціон характеризувався вартістю продажу одиниці з партії квітів як вихідною (цільовою) ознакою та значеннями наступних вхідних ознак:

- кількість учасників аукціону;
- час початку аукціону в секундах;
- відповідність першому типу бегоній;
- відповідність другому типу бегоній;
- здійснення продажу в понеділок;
- здійснення продажу у вівторок;
- здійснення продажу в середу;
- здійснення продажу в четвер.

Зі всіх даних 40 % відносились до тестувальної вибірки та 60 % для навчальної. Розподіл між вибірками відбувався випадково. Значення вартості продажу, що є цільовою ознакою, нормувались перед передачею в модель для навчання, тобто зводились до інтервалу від 0 до 1.

Модель на основі дерева рішень реалізовувалась через алгоритм машинного навчання CART, що реалізований у пакеті scikit-learn. Дерево створювалось без обмеження на його глибину. Навчання реалізовувалось на основі середньоквадратичної похибки або Mean Squared Error (MSE).

Для оцінювання всіх створених моделей виконувались наступні оцінки:

- середньоквадратична похибка або MSE;
- середня абсолютна похибка або Mean Absolute Error (MAE);

– коренева середньоквадратична похибка або Root Mean Squared Error (RMSE).

Для запропонованого методу було розглянуто декілька варіацій моделі, створеної на основі методу Random Forest. Було розглянуто різну кількість дерев у ансамблі, включаючи:

- 700 дерев;
- 750 дерев;
- 900 дерев;
- 1000 дерев;
- 600 дерев;
- 500 дерев;
- 300 дерев;
- 100 дерев;
- 50 дерев (табл. 5.1).

Таблиця 5.1 – Результати прогнозування вартості постачання живих квітів на основі варіацій запропонованого методу

Кількість дерев у ансамблі Random Forest	MAE	MSE	RMSE
50	0,0246055	0,0010694	0,0327014
100	0,0246873	0,0010159	0,0318732
300	0,0238526	0,0010118	0,0318085
500	0,0234413	0,0009583	0,0309558
600	0,0239213	0,0010009	0,0316373
700	0,0242298	0,00101	0,0317807
750	0,0236311	0,0009766	0,0312502
900	0,0235497	0,0009705	0,0311536
1000	0,0238729	0,001003	0,0316706

Обмеження на глибину дерева не вводились. Реалізовувалась оцінка узагальнюючих властивостей моделі на основі коефіцієнта детермінації.

Найкращий результат за всіма 3 показниками дозволила отримати модель, побудована як ансамбль з 500 дерев.

Як ще одна альтернатива була розглянута модель регресії, побудована основі методу опорних векторів (табл. 5.2).

Таблиця 5.2 – Підсумкові результати прогнозування вартості постачання живих квітів

Спосіб створення моделі	MAE	MSE	RMSE
Метод опорних векторів	0,0453399	0,0027406	0,0523512
CART	0,0262265	0,0016424	0,040526
Запропонований метод прогнозування вартості постачання живих квітів	0,0234413	0,0009583	0,0309558

Запропонований метод прогнозування вартості постачання живих квітів, що використовував модель на основі методу Random Forest, дозволив зменшити рівень MAE на 48,3 % порівняно з методом опорних векторів та на 10,6 % порівняно з деревами рішень CART. У свою чергу показник MSE був зменшений на 65 % порівняно з методом опорних векторів та на 41,65 % порівняно з деревами рішень CART. Рівень похибки RMSE був зменшений запропонованим методом на 40,87 % порівняно з методом опорних векторів та на 23,6 % порівняно з деревами рішень CART. Тож запропонований метод прогнозування вартості постачання живих квітів продемонстрував найкращий результат прогнозування.

5.4 Висновки за розділом 5

У розділі 5 представлено результати тестування програмного забезпечення на основі перевірки різних сценаріїв роботи користувачів з програмою та експериментального дослідження запропонованого методу прогнозування вартості постачання живих квітів на основі вибірки даних, в якій представлені результати проведення аукціону з продажу квітів. Запропонований метод прогнозування вартості постачання живих квітів було порівняно з моделями, створеними на основі методу опорних векторів та дерев рішень. За усіма показниками (MAE, MSE, RMSE) запропонований метод дозволив досягти меншої похибки результатів прогнозування на тестовій вибірці даних.

ВИСНОВКИ

За підсумками виконання роботи розроблено програмне забезпечення продажу живих квітів для підвищення зручності роботи користувачів на основі інтеграції прогнозування вартості постачання квітів.

Створений вебзастосунок продажу живих квітів дозволяє виконувати прогнозування вартості постачання живих квітів, враховуючи це при формуванні очікуваної вартості продажу квітів для гуртових та індивідуальних покупців, формувати замовлення букетів та сортів квітів, надає засоби для підтримки знаходження потрібних квітів та букетів, дозволяє висловлювати власні оцінки стосовно придбаних живих квітів та букетів. Розробку виконано мовою Python як основною мовою проєкту та з використанням засобів веброзробки, включаючи фреймворк Django, бібліотеку Bootstrap, мови HTML, CSS, JavaScript, систему керування базами даних PostgreSQL.

У роботі описано основні етапи над створенням цього вебзастосунку, включаючи аналіз аналогів, до яких зокрема віднесено Flowers.ua та Dicentra, визначення переваг програми відносно цих аналогів, визначення засобів розробки, моделювання сценаріїв роботи представника магазину живих квітів та покупця з програмою, визначення структури бази даних програми продажу живих квітів, описано логіку створених програмних функцій, описано логіку процесу роботи з програмою.

Наукова новизна роботи полягає в методі прогнозування вартості постачання живих квітів, який визначає створення і навчання моделі ансамблю дерев рішень на основі методу Random Forest, що дозволяє визначити вартість продажу одиниці квітів на аукціоні. Експериментальне дослідження підтвердило зменшення похибки прогнозування вартості живих квітів порівняно з методом опорних векторів та окремим деревом рішень на основі алгоритму CART.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Доставка квітів по Україні та світу [Електрон. ресурс] : замовте букет квітів з доставкою по Україні на Flowers.ua. – Режим доступу : <https://flowers.ua/>.
2. Доставка квітів у Запоріжжі – букети від ProFlowers.ua [Електрон. ресурс]. – Режим доступу : <https://proflowers.ua/ua/dostavka-cvetov-zaporozhye>.
3. Купити квіти з доставкою в Києві – інтернет магазин квітів Dicentra [Електрон. ресурс]. – Режим доступу <https://dicentra.ua/>.
4. Flower Auction [Electronic resource]. – Access mode : <https://www.kaggle.com/datasets/davidgauthier/flowersauction/data>.
5. 10 popular regression algorithms in Machine Learning [Electronic resource]. – Access mode : <https://www.onlinemanipal.com/blogs/popular-regression-algorithms-in-machine-learning>.
6. Verma, N. An Introduction to Support Vector Regression (SVR) in Machine Learning [Electronic resource] / N. Verma. – Access mode : <https://medium.com/@nandiniverma78988/an-introduction-to-support-vector-regression-svr-in-machine-learning-681d541a829a>.
7. Unveiling Decision Tree Regression: Exploring its Principles, Implementation [Electronic resource]. – Access mode : <https://medium.com/@vk.viswa/unveiling-decision-tree-regression-exploring-its-principles-implementation-beb882d756c6#:~:text=Decision%20tree%20regression%20is%20a%20machine%20learning%20technique%20that%20constructs,focuses%20on%20estimating%20numeric%20outcomes>.
8. Chakure, A. Random Forest Regression in Python Explained [Electronic resource] / A. Chakure. – Access mode : <https://builtin.com/data-science/random-forest-python#:~:text=What%20Is%20Random%20Forest%20Regression,trees%20while%20building%20the%20trees..>
9. Blog Tool, Publishing Platform, and CMS – WordPress.org [Electronic resource]. – Access mode : <https://wordpress.org/>.

10. Django [Electronic resource]. – Access mode : <https://github.com/django/django>.
11. Rent4u Free Website Template. Free CSS Templates. Free CSS [Electronic resource]. – Access mode : <https://www.free-css.com/free-css-templates/page294/rent4u>.

ДОДАТОК А
Технічне завдання

Вступ

Робота магазину живих квітів пов'язана з продажем квітів, які при цьому нерідко поставляються з-за меж країни, тобто від постачальників, з якими магазин нерідко безпосередньо не контактує. Одним з таких варіантів є придбання квітів на спеціальних аукціонах закордоном, коли потім квіти доставляються в потрібну країну, а далі продаються в магазинах. Не кожен магазин працює з такими аукціонами, але для великого магазину це є можливим.

Саме на реалізацію підтримки роботи такого магазину і направлена робота програмного застосунку, який потрібно створити в цій роботі. При цьому корисним для користувача в такому випадку є розуміння вартості, за яку він зможе купити квіти, які будуть доставлені в майбутній період часу. Це може дозволити планувати йому свої покупки, а з іншого боку розуміти, якою є очікувана вартість на певні періоди часу. Для цього корисним є наявність функціонально механізму прогнозування вартості постачання живих квітів до магазину. Саме реалізація такого механізму в складі програмного застосунку і є важливим завданням цієї роботи.

A.1 Підстави для розробки

На підставі наказу № 398 від 7 жовтня 2024 р., який був виданий у Національному університеті «Запорізька політехніка», де і ведеться розробка в межах даної дипломної кваліфікаційної роботи магістра, визначена тема роботи «Дослідження та програмна реалізація методів прогнозування під час постачання живих квітів», що було деталізовано також виданим завданням на роботу.

A.2 Призначення розробки

Програмне забезпечення продажу живих квітів повинно надавати можливість відбирати потрібні букети або сорти живих квітів, замовляти такі букети або квіти індивідуально, керувати цими замовленнями зокрема зі сторони магазину, визначати наявність та цілісні постачання живих квітів від зовнішніх постачальників через придбання квітів на аукціоні. Робота має підтримуватися як для індивідуальних покупців, так і для великих гуртових покупців.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Для реалізації комплексної роботи вебзастосунку продажу живих квітів як щодо продажу і замовлення живих квітів, так і щодо роботи з постачаннями живих квітів необхідно забезпечити такі функціональні можливості:

- відбір потрібного букету живих квітів;
- відбір індивідуальних живих квітів;
- замовлення живих квітів;
- змінювання стану замовлення живих квітів;
- додавання букету живих квітів або сортів квітів до бажання та вилучення раніше доданого букету або сорту квітів;
- перегляд бажаних букетів або сортів живих квітів;
- оцінювання букету або сорту живих квітів;
- перегляд оцінених користувачем букетів або сортів живих квітів;
- прогнозування вартості постачання живих квітів;
- додавання нового постачання живих квітів;
- редагування даних постачання живих квітів;
- перегляд запланованих та виконаних постачань живих квітів;
- визначення наявності сортів живих квітів та букетів;

- перегляд виконаних користувачем замовлень живих квітів;
- перегляд замовлень живих квітів у магазині;
- пошук замовлення живих квітів;
- додавання сортів живих квітів;
- перегляд інформації про букет живих квітів;
- перегляд інформації про сорт живих квітів;
- пошук серед сортів живих квітів;
- пошук серед букетів живих квітів;
- перегляд інформації про замовлення живих квітів;
- перегляд інформації про користувача;
- редагування інформації про букет живих квітів;
- редагування інформації про сорт живих квітів;
- редагування інформації про користувача;
- додавання інформації про букет живих квітів;
- авторизація покупців та представників магазину живих квітів;
- реєстрація покупців та представників магазину живих квітів.

A.3.2 Умови експлуатації

Експлуатація програмного забезпечення продажу живих квітів має забезпечуватися виконанням ряду вимог до серверної частини:

- забезпечення вебсерверу з підтримкою інтерпретатора мови програмування Python 3;
- налаштування Django-застосунку для виконання на сервері з реалізацією міграції для забезпечення необхідної структури бази даних за допомогою підтримки системи керування базами даних PostgreSQL;
- інсталювані Python-пакети Numpy, Pandas, Scikit-learn.

А.3.3 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення продажу живих квітів має забезпечуватися серверною частиною з виконанням таких мінімальних параметрів технічних засобів:

- двоядерний процесор;
- 4 Гб оперативної пам'яті;
- 10 Гб простору на жорсткому диску, що необхідно для збереження даних самого вебсайту та фотографій живих квітів, файлів вебзастосунку, бази даних: в залежності від якості, кількості фотографій можливе подальше збільшення мінімальних вимог.

А.4 Порядок контролю та приймання

Робота має виконуватися у відповідності з затвердженим завданням і контролюється керівником. Отримані результати роботи представляються в пояснювальній записці.

Завершальним етапом виконання та відповідно контролю і приймання дипломної кваліфікаційної роботи магістра є її захист, який виконується перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

```

from django.shortcuts import render
from .models import Flowerkind
from .models import Flowersort
from .models import FlowersortvariantPackaging
from .models import FlowershopUser
from .models import Flowersortvariant
from .models import FlowersortvariantPhoto
from .models import Flowersortvariantsupply
from .models import Purpose
from .models import Bouquet
from django.db.models import Q
from .models import BouquetPurpose
from .models import BouquetPackaging
from .models import BouquetFlower
from .models import BouquetPhoto
from .models import FlowershopOrder
from .models import FlowershopOrderStatus
from .models import FlowershopOrderBouquet
from .models import FlowershopOrderFlowersortvariant
from .models import Flowersortvariantscore
from .models import Bouquetscore
from .models import Packaging
from django.core.paginator import Paginator
from datetime import datetime

def choose_flower_view(request):
    flowers =
    Flowersortvariant.objects.select_related("flowersort",
    "flowersort__flowerkind", "basephoto", "basesupply").all()
    flower_kinds = request.GET.get("kinds")
    if flower_kinds:
        flower_kinds_to_search = []
        flower_kinds_separated = flower_kinds.split("@")
        for flower_kind in flower_kinds_separated:
            flower_kind = int(flower_kind)
            flower_kind = Flowerkind.objects.filter(id =
flower_kind)

            flower_kind = flower_kind.first()
            flower_kinds_to_search.append(flower_kind)
        flowers = flowers.filter(flowersort__flowerkind__in
= flower_kinds_to_search)
    flower_sorts = request.GET.get("sorts")
    if flower_sorts:
        flower_sorts_to_search = []
        flower_sorts_separated = flower_sorts.split("@")
        for flower_sort in flower_sorts_separated:
            flower_sort = int(flower_sort)
            flower_sort = Flowersort.objects.filter(id =
flower_sort)

            flower_sort = flower_sort.first()
            flower_sorts_to_search.append(flower_sort)

```

```

        flowers = flowers.filter(flowersort__in =
flower_sorts_to_search)
        flower_sizes = request.GET.get("sizes")
        if flower_sizes:
            flower_sizes_to_search = []
            flower_sizes_separated = flower_sizes.split("@")
            for flower_size in flower_sizes_separated:
                flower_size = int(flower_size)
                flower_sizes_to_search.append(flower_size)
            flowers = flowers.filter(size__in =
flower_sizes_to_search)
        flower_colors = request.GET.get("colors")
        if flower_colors:
            flower_colors_to_search = []
            flower_colors_separated = flower_colors.split("@")
            for flower_color in flower_colors_separated:
                flower_color = int(flower_color)
                flower_colors_to_search.append(flower_color)
            flowers = flowers.filter(color__in =
flower_colors_to_search)
        flower_maxheight = request.GET.get("maxheight")
        if flower_maxheight:
            flower_maxheight = float(flower_maxheight)
            flowers = flowers.filter(height__lte =
flower_maxheight)
        flower_minheight = request.GET.get("minheight")
        if flower_minheight:
            flower_minheight = float(flower_minheight)
            flowers = flowers.filter(height__gte =
flower_minheight)
        flower_maxscore = request.GET.get("maxscore")
        if flower_maxscore:
            flower_maxscore = float(flower_maxscore)
            flowers = flowers.filter(score__lte =
flower_maxscore)
        flower_minscore = request.GET.get("minscore")
        if flower_minscore:
            flower_minscore = float(flower_minscore)
            flowers = flowers.filter(score__gte =
flower_minscore)
        flower_maxprice = request.GET.get("maxprice")
        if flower_maxprice:
            flower_maxprice = float(flower_maxprice)
        flower_minprice = request.GET.get("minprice")
        if flower_minprice:
            flower_minprice = float(flower_minprice)
        flowers_results = []
        for flower in flowers:
            leave = True
            if flower.basesupply.leftnumber > 0:
                price = flower.basesupply.fixedprice
                if price == 0:
                    price = flower.basesupply.plannedprice

```

```

        if price == 0:
            price = flower.basesupply.forecastprice
* (1 + flower.basesupply.addpercentprice) +
flower.basesupply.addprice
        if flower_maxprice and price > flower_maxprice:
            leave = False
        if flower_minprice and price < flower_minprice:
            leave = False
        flower.detprice = price
    else:
        flower_supplies =
Flowersortvariantsupply.objects.filter(flowersortvariant =
flower, supplydate__gt = flower.basesupply.supplydate,
leftnumber__gt = 0)
        flower_supplies =
flower_supplies.order_by("supplydate")
        if not flower_supplies:
            leave = False
        flower_supply = flower_supplies.first()
        price = flower_supply.fixedprice
        if price == 0:
            price = flower_supply.plannedprice
            if price == 0:
                price = flower_supply.forecastprice *
(1 + flower_supply.addpercentprice) + flower_supply.addprice
            flower.detprice = price
            if flower_maxprice and price > flower_maxprice:
                leave = False
            if flower_minprice and price < flower_minprice:
                leave = False
        if leave:
            flowers_results.append(flower)
flower_order = request.GET.get("order")
if flower_order:
    flower_order = int(flower_order)
    if flower_order == 1:
        flowers_results = sorted(flowers_results,
key=lambda x: x.detprice, reverse=True)
    elif flower_order == 2:
        flowers_results = sorted(flowers_results,
key=lambda x: x.detprice, reverse=False)
    elif flower_order == 3:
        flowers_results = sorted(flowers_results,
key=lambda x: x.score, reverse=True)
    elif flower_order == 4:
        flowers_results = sorted(flowers_results,
key=lambda x: x.score, reverse=False)
    elif flower_order == 5:
        flowers_results = sorted(flowers_results,
key=lambda x: x.height, reverse=True)
    elif flower_order == 6:
        flowers_results = sorted(flowers_results,
key=lambda x: x.height, reverse=False)

```

```

        elif flower_order == 7:
            flowers_results = sorted(flowers_results,
key=lambda x: x.flowersort.sortname, reverse=True)
        elif flower_order == 8:
            flowers_results = sorted(flowers_results,
key=lambda x: x.flowersort.sortname, reverse=False)
        flower_interval = request.GET.get("variant")
        if not flower_interval:
            flower_ids = []
            for flowers_result in flowers_results:
                if flowers_result.flowersort not in flower_ids:
                    flowersort = flowers_result.flowersort
                    flower_ids.append(flowersort)
        else:
            flower_ids = None
            flower_paginator = Paginator(flowers_results, 10)
            flower_interval = request.GET.get("page")
            flower_objects =
flower_paginator.get_page(flower_interval)
            flower_context = {}
            flower_context["flowers"] = flower_objects
            flower_context["sorts"] = flower_ids
            kinds = Flowerkind.objects.all()
            sorts = Flowersort.objects.all()
            flower_context["kinds"] = kinds
            flower_context["allsorts"] = sorts
            return render(request, "flowerspl/flowerchoose.html",
flower_context)

def choose_bouquet_view(request):
    bouquets =
Bouquet.objects.select_related("basephoto").all()
    flower_kinds = request.GET.get("kinds")
    if flower_kinds:
        flower_kinds_to_search = []
        flower_kinds_separated = flower_kinds.split("@")
        for flower_kind in flower_kinds_separated:
            flower_kind = int(flower_kind)
            flower_kind = Flowerkind.objects.filter(id =
flower_kind)

            flower_kind = flower_kind.first()
            flower_kinds_to_search.append(flower_kind)
        bouque_flowers =
BouquetFlower.objects.select_related("bouquet").filter(flowersor
tvariant__flowerkind__in = flower_kinds_to_search)
        to_bouques = bouque_flowers.values("bouquet__id")
        bouquets = bouquets.filter(id__in = to_bouques)
        flower_sorts = request.GET.get("sorts")
        if flower_sorts:
            flower_sorts_to_search = []
            flower_sorts_separated = flower_sorts.split("@")
            for flower_sort in flower_sorts_separated:
                flower_sort = int(flower_sort)

```

```

        flower_sort = Flowersort.objects.filter(id =
flower_sort)

        flower_sort = flower_sort.first()
        flower_sorts_to_search.append(flower_sort)
        bouque_flowers =
BouquetFlower.objects.select_related("bouquet").filter(flowersor
tvariant__in = flower_sorts_to_search)
        to_bouques = bouque_flowers.values("bouquet__id")
        bouquets = bouquets.filter(id__in = to_bouques)
        bouquet_purposes = request.GET.get("purposes")
        if bouquet_purposes:
            bouquet_purposes_to_search = []
            bouquet_purposes_separated =
bouquet_purposes.split("@")
            for bouquet_purposein bouquet_purposes_separated:
                bouquet_purpose = int(bouquet_purpose)

bouquet_purposes_to_search.append(bouquet_purpose)
            bouque_flowers =
BouquetPurpose.objects.select_related("bouquet",
"purpose").filter(purpose__id__in = bouquet_purposes_to_search)
            to_bouques = bouque_flowers.values("bouquet__id")
            bouquets = bouquets.filter(id__in = to_bouques)
            flower_sizes = request.GET.get("sizes")
            if flower_sizes:
                flower_sizes_to_search = []
                flower_sizes_separated = flower_sizes.split("@")
                for flower_size in flower_sizes_separated:
                    flower_size = int(flower_size)
                    flower_sizes_to_search.append(flower_size)
                flowers = Flowersortvariant.objects.filter(size__in
= flower_sizes_to_search)
                bouque_flowers =
BouquetFlower.objects.select_related("bouquet").filter(flowersor
tvariant__in = flowers)
                to_bouques = bouque_flowers.values("bouquet__id")
                bouquets = bouquets.filter(id__in = to_bouques)
                flower_colors = request.GET.get("colors")
                if flower_colors:
                    flower_colors_to_search = []
                    flower_colors_separated = flower_colors.split("@")
                    for flower_color in flower_colors_separated:
                        flower_color = int(flower_color)
                        flower_colors_to_search.append(flower_color)
                    flowers =
Flowersortvariant.objects.filter(color__in =
flower_colors_to_search)
                    bouque_flowers =
BouquetFlower.objects.select_related("bouquet").filter(flowersor
tvariant__in = flowers)
                    to_bouques = bouque_flowers.values("bouquet__id")
                    bouquets = bouquets.filter(id__in = to_bouques)
                    flower_maxheight = request.GET.get("maxheight")

```

```

        if flower_maxheight:
            flower_maxheight = float(flower_maxheight)
            flowers = Flowersortvariant.objects.filter
(height__lte = flower_maxheight)
            bouquet_flowers =
BouquetFlower.objects.select_related("bouquet").filter(flowersor
tvariant__in = flowers)
            to_bouques = bouquet_flowers.values("bouquet__id")
            bouquets = bouquets.filter(id__in = to_bouques)
            flower_minheight = request.GET.get("minheight")
            if flower_minheight:
                flower_minheight = float(flower_minheight)
                flowers =
Flowersortvariant.objects.filter(height__gte = flower_maxheight)
                bouquet_flowers =
BouquetFlower.objects.select_related("bouquet").filter(flowersor
tvariant__in = flowers)
                to_bouques = bouquet_flowers.values("bouquet__id")
                bouquets = bouquets.filter(id__in = to_bouques)
                bouquet_maxscore = request.GET.get("maxscore")
                if bouquet_maxscore:
                    bouquet_maxscore = float(bouquet_maxscore)
                    bouquets = bouquets.filter(score__lte =
bouquet_maxscore)
                bouquet_minscore = request.GET.get("minscore")
                if bouquet_minscore:
                    bouquet_minscore = float(bouquet_minscore)
                    bouquets = bouquets.filter(score__gte =
bouquet_minscore)
                bouquet_maxprice = request.GET.get("maxprice")
                if bouquet_maxprice:
                    bouquet_maxprice = float(bouquet_maxprice)
                    bouquets = bouquets.filter(Q(fixedprice__lte =
bouquet_maxprice) | Q(currentfixedprice__gt = 0,
currentfixedprice__lte = bouquet_maxprice))
                bouquet_minprice = request.GET.get("minprice")
                if bouquet_minprice:
                    bouquet_minprice = float(bouquet_minprice)
                    bouquets = bouquets.filter(Q(fixedprice__gte =
bouquet_minprice) | Q(currentfixedprice__gt = 0,
currentfixedprice__gte = bouquet_minprice))
                bouquet_results = []
                bouquet_order = request.GET.get("order")
                if bouquet_order:
                    bouquet_order = int(bouquet_order)
                    if bouquet_order == 1:
                        bouquet_results = bouquets.order_by("-
fixedprice")
                    elif bouquet_order == 2:
                        bouquet_results =
bouquets.order_by("fixedprice")
                    elif bouquet_order == 3:
                        bouquet_results = bouquets.order_by("-score")

```

```

        elif bouquet_order == 4:
            bouquet_results = bouquets.order_by("score")
        elif bouquet_order == 5:
            bouquet_results = bouquets.order_by("caption")
        elif bouquet_order == 6:
            bouquet_results = bouquets.order_by("-caption")
    bouquet_paginator = Paginator(bouquet_results, 10)
    bouquet_interval = request.GET.get("page")
    bouquet_objects =
bouquet_paginator.get_page(bouquet_interval)
    bouquet_context = {}
    bouquet_context["bouquets"] = bouquet_objects
    kinds = Flowerkind.objects.all()
    sorts = Flowersort.objects.all()
    bouquet_context["kinds"] = kinds
    bouquet_context["allsorts"] = sorts
    return render(request, "flowerspl/bouquetchoose.html",
bouquet_context)

```

```

    def flower_desire_add_view(request, flowerorbouquet,
flower_id):
        if request.user.is_authenticated:
            flower_user = request.user
            flower_shop_users =
FlowershopUser.objects.prefetch_related("flowersortvariants",
"bouquets").select_related("djangouser")
            flower_shop_users =
flower_shop_users.filter(djangouser = flower_user)
            flower_shop_user = flower_shop_users.first()
            if not flowerorbouquet:
                bouquet = Bouquet.objects.filter(id =
flower_id)

                if bouquet in flower_shop_users.bouquets:
                    flower_shop_users.bouquets.remove(bouquet)
                    flower_shop_users.save()
                else:
                    flower_shop_users.bouquets.add(bouquet)
                    flower_shop_users.save()
            else:
                flowersortvariant =
Flowersortvariant.objects.filter(id = flower_id)
                if flowersortvariant in
flower_shop_users.flowersortvariants:

flower_shop_users.flowersortvariants.remove(flowersortvariant)
                    flower_shop_users.save()
                else:

flower_shop_users.flowersortvariants.add(flowersortvariant)
                    flower_shop_users.save()

```

```

def flower_desires_view(request):
    if request.user.is_authenticated:

```

```

        flower_user = request.user
        page_number = request.GET.get("flowerorbouquet")
        flowerorbouquet = int(flowerorbouquet)
        if flowerorbouquet == 0:
            flower_shop_users =
FlowershopUser.objects.prefetch_related("flowersortvariants",
"flowersortvariants__flowersort",
"flowersortvariants__flowersort__flowerkind",
"flowersortvariants__basephoto").select_related("djangouser")
            flower_shop_users =
flower_shop_users.filter(djangouser = flower_user)
            flower_shop_user = flower_shop_users.first()
            flower_paginator =
Paginator(flower_shop_user.flowersortvariants, 10)
            flower_interval = request.GET.get("page")
            flower_objects =
flower_paginator.get_page(flower_interval)
            return render(request,
"flowerspl/desires.html", {"objects": bouquet_objects })
        else:
            flower_shop_users =
FlowershopUser.objects.prefetch_related("bouquets",
"bouquets__basephoto").select_related("djangouser")
            flower_shop_users =
flower_shop_users.filter(djangouser = flower_user)
            flower_shop_user = flower_shop_users.first()
            bouquet_paginator =
Paginator(flower_shop_user.bouquets, 10)
            bouquet_interval = request.GET.get("page")
            bouquet_objects =
bouquet_paginator.get_page(bouquet_interval)
            return render(request,
"flowerspl/desires.html", {"objects": bouquet_objects })
        else:
            return render(request, "flowerspl/desires.html",
{"objects": None })

def flower_scores_view(request):
    flowerorbouquet = request.GET.get("flowerorbouquet")
    minscore = request.GET.get("minscore")
    if minscore:
        minscore = float(minscore)
    maxscore = request.GET.get("maxscore")
    if maxscore:
        maxscore = float(maxscore)
    if request.user.is_authenticated:
        flower_user = request.user
        flower_shop_users =
FlowershopUser.objects.select_related("djangouser")
        flower_shop_users =
flower_shop_users.filter(djangouser = flower_user)
        flower_shop_user = flower_shop_users.first()

```

```

        bouquet_scores =
Bouquetscore.objects.select_related("bouquet").filter(flowershop
user = flower_shop_user)
        if minscore:
            bouquet_scores =
bouquet_scores.filter(score__gte = minscore)
            bouquet_scores = bouquet_scores.order_by("-
scoredat")
        if maxscore:
            bouquet_scores =
bouquet_scores.filter(score__lte = maxscore)
            bouquet_scores = bouquet_scores.order_by("-
scoredat")
        flowersort_scores =
Flowersortvariantscore.objects.select_related("flowersortvariant
").filter(flowershopuser = flower_shop_user)
        if minscore:
            flowersort_scores =
flowersort_scores.filter(score__gte = minscore)
            flowersort_scores =
flowersort_scores.order_by("-scoredat")
        if maxscore:
            flowersort_scores =
flowersort_scores.filter(score__lte = maxscore)
            flowersort_scores =
flowersort_scores.order_by("-scoredat")
            flowersort_scores = flowersort_scores.order_by("-
scoredat")
        if flowerorbouquet == 0:
            flower_paginator = Paginator(flowersort_scores,
10)
            flower_interval = request.GET.get("page")
            flower_objects =
flower_paginator.get_page(flower_interval)
            return render(request, "flowerspl/scores.html",
{"objects": flower_objects, "object_types": "flowers" })
        elif flowerorbouquet == 1:
            bouquet_paginator = Paginator(bouquet_scores,
10)
            bouquet_interval = request.GET.get("page")
            bouquet_objects =
bouquet_paginator.get_page(bouquet_interval)
            return render(request, "flowerspl/scores.html",
{"objects": bouquet_objects, "object_types": "bouquets" })
        elif flowerorbouquet == 2:
            all_scores = flowersort_scores.values() +
bouquet_scores.values()
            all_scores.sort(key=lambda x: x.scoredat,
reverse=True)
            bouquet_paginator = Paginator(all_scores, 10)
            bouquet_interval = request.GET.get("page")
            bouquet_objects =
bouquet_paginator.get_page(bouquet_interval)

```

```

        flower_context = {}
        flower_context["objects"] = bouquet_objects
        flower_context["object_types"] = "both"
        return render(request, "flowerspl/scores.html",
flower_context)

    def flowershop_user_orders_view(request):
        if request.user.is_authenticated:
            flowershop_order_status = request.GET.get("status")
            flowershop_order_status =
int(flowershop_order_status)
            flower_user = request.user
            flower_shop_users =
FlowershopUser.objects.select_related("djangouser")
            flower_shop_users =
flower_shop_users.filter(djangouser = flower_user)
            flower_shop_user = flower_shop_users.first()
            flowershop_orders =
FlowershopOrder.objects.filter(flowershopUser =
flower_shop_user)
            if flowershop_order_status != -1:
                flowershop_orders =
FlowershopOrder.objects.filter(currentstatus =
flowershop_order_status)
            flowershop_orders = flowershop_orders.order_by("-
createdat")
            flowershop_paginator = Paginator(flowershop_orders,
10)
            flowershop_interval = request.GET.get("page")
            flowershop_objects =
flowershop_paginator.get_page(flowershop_interval)
            flower_context = {}
            flower_context["objects"] = flowershop_objects
            return render(request,
"flowerspl/myflowerorders.html", flower_context)

    def bouquet_view(request, bouquet_id):
        bouquet =
Bouquet.objects.select_related("basephoto").filter(id =
bouquet_id)
        bouquet.desired = False
        bouquet.scored = False
        purposes =
BouquetPurpose.objects.select_related("purpose ").filter(bouquet
= bouquet)
        packagings =
BouquetPackaging.objects.select_related("packaging").filter(bouq
uet = bouquet)
        flowers =
BouquetFlower.objects.select_related("flowersortvariant",
"flowersortvariant__flowersort",
"flowersortvariant__flowersort__flowerkind").filter(bouquet =
bouquet)

```

```

        for flower in flowers:
            flower.supply =
Flowersortvariant.supply.objects.filter(flowersortvariant =
flower.flowersortvariant, leftnumber__gt = 0, status__lt = 5)
            flower.supply =
flower.supply.order_by("buydatetime")
            photos = BouquetPhoto.objects.filter(bouquet = bouquet)
            bouquet_scores = Bouquetscore.objects.filter(bouquet =
bouquet)
            bouquet_scores = bouquet_scores.order_by("-scoredat")
            if request.user.is_authenticated:
                flower_user = request.user
                flower_shop_users =
FlowershopUser.objects.prefetch_related("bouquets").select_relat
ed("djangouser")
                flower_shop_users =
flower_shop_users.filter(djangouser = flower_user)
                flower_shop_user = flower_shop_users.first()
                if bouquet in flower_shop_user.bouquets:
                    bouquet.desired = True
                    user_bouquet_score =
Bouquetscore.objects.filter(bouquet = bouquet, flowershopuser =
flower_shop_user)
                    if user_bouquet_score:
                        bouquet.scored = True
                flower_context = {}
                flower_context["bouquet"] = bouquet
                flower_context["purposes"] = purposes
                flower_context["packagings"] = packagings
                flower_context["flowers"] = flowers
                flower_context["photos"] = photos
                flower_context["scores"] = bouquet_scores
                return render(request, "flowerspl/bouquet.html",
flower_context)

def flower_view(request, flower_id):
    flower_sort_variant =
Flowersortvariant.objects.select_related("basephoto").select_rel
ated("flowersort").select_related("flowersort__flowerkind").filt
er(id = flower_id)
    flower_sort_variant.desired = False
    flower_sort_variant.scored = False
    packagings =
FlowersortvariantPackaging.objects.select_related("packaging").f
ilter(flowersortvariant = flower_sort_variant)
    photos =
FlowersortvariantPhoto.objects.filter(flowersortvariant =
flower_sort_variant)
    flower_sort_variant_scores =
Flowersortvariantscore.objects.filter(flowersortvariant =
flower_sort_variant)
    flower_sort_variant_scores =
flower_sort_variant_scores.order_by("-scoredat")

```

```

        supplies =
Flowersortvariantsupply.objects.filter(flowersortvariant =
flower_sort_variant, leftnumber__gt = 0, status__lt = 5)
        supplies = supplies.order_by("buydatetime")
        if request.user.is_authenticated:
            flower_user = request.user
            flower_shop_users =
FlowershopUser.objects.prefetch_related("flowersortvariants").se
lect_related("djangouser")
            flower_shop_users =
flower_shop_users.filter(djangouser = flower_user)
            flower_shop_user = flower_shop_users.first()
            if flower_sort_variant in
flower_shop_user.flowersortvariants:
                flower_sort_variant.desired = True
                user_flower_score =
Flowersortvariantscore.objects.filter(flowersortvariant =
flower_sort_variant, flowershopuser = flower_shop_user)
                if user_flower_score:
                    flower_sort_variant.scored = True
            flower_context = {}
            flower_context["flower"] = flower_sort_variant
            flower_context["packagings"] = packagings
            flower_context["photos"] = photos
            flower_context["scores"] = flower_sort_variant_scores
            flower_context["supplies"] = supplies
            return render(request,
"flowerspl/flowersortvariant.html", flower_context)

def search_flower_view(request):
    if not request.user.is_authenticated:
        bouquet_context = {}
        bouquet_context["bouquets"] = None
        return render(request,
"flowerspl/flowersearch.html", bouquet_context)
    else:
        flower_user = request.user
        if not flower_user.is_staff:
            bouquet_context = {}
            bouquet_context["bouquets"] = None
            return render(request,
"flowerspl/flowersearch.html", bouquet_context)
        flowers =
Flowersortvariant.objects.select_related("flowersort",
"flowersort__flowerkind", "basephoto", "basesupply").all()
        flower_variant_id = request.GET.get("variantid")
        if flower_variant_id:
            flower_variant_id = int(flower_variant_id)
            flowers = flowers.filter(id = flower_variant_id)
        flower_search_sort = request.GET.get("flowersort")
        if flower_search_sort:

```

```

        flower_search_sorts =
Flowersort.objects.select_related("basephoto",
"flowerkind").filter(sortname__icontains = flower_search_sort)
        flowers = flowers.filter(flowersort__in =
flower_search_sorts)
        flower_kinds = request.GET.get("kinds")
        if flower_kinds:
            flower_kinds_to_search = []
            flower_kinds_separated = flower_kinds.split("@")
            for flower_kind in flower_kinds_separated:
                flower_kind = int(flower_kind)
                flower_kind = Flowerkind.objects.filter(id =
flower_kind)
                flower_kind = flower_kind.first()
                flower_kinds_to_search.append(flower_kind)
            flowers = flowers.filter(flowersort__flowerkind__in
= flower_kinds_to_search)
        flower_sorts = request.GET.get("sorts")
        if flower_sorts:
            flower_sorts_to_search = []
            flower_sorts_separated = flower_sorts.split("@")
            for flower_sort in flower_sorts_separated:
                flower_sort = int(flower_sort)
                flower_sort = Flowersort.objects.filter(id =
flower_sort)
                flower_sort = flower_sort.first()
                flower_sorts_to_search.append(flower_sort)
            flowers = flowers.filter(flowersort__in =
flower_sorts_to_search)
        flower_sizes = request.GET.get("sizes")
        if flower_sizes:
            flower_sizes_to_search = []
            flower_sizes_separated = flower_sizes.split("@")
            for flower_size in flower_sizes_separated:
                flower_size = int(flower_size)
                flower_sizes_to_search.append(flower_size)
            flowers = flowers.filter(size__in =
flower_sizes_to_search)
        flower_colors = request.GET.get("colors")
        if flower_colors:
            flower_colors_to_search = []
            flower_colors_separated = flower_colors.split("@")
            for flower_color in flower_colors_separated:
                flower_color = int(flower_color)
                flower_colors_to_search.append(flower_color)
            flowers = flowers.filter(color__in =
flower_colors_to_search)
        flower_maxheight = request.GET.get("maxheight")
        if flower_maxheight:
            flower_maxheight = float(flower_maxheight)
            flowers = flowers.filter(height__lte =
flower_maxheight)
        flower_minheight = request.GET.get("minheight")

```

```

        if flower_minheight:
            flower_minheight = float(flower_minheight)
            flowers = flowers.filter(height__gte =
flower_maxheight)
            flower_maxscore = request.GET.get("maxscore")
            if flower_maxscore:
                flower_maxscore = float(flower_maxscore)
                flowers = flowers.filter(score__lte =
flower_maxscore)
            flower_minscore = request.GET.get("minscore")
            if flower_minscore:
                flower_minscore = float(flower_minscore)
                flowers = flowers.filter(score__gte =
flower_minscore)
            flower_maxprice = request.GET.get("maxprice")
            if flower_maxprice:
                flower_maxprice = float(flower_maxprice)
            flower_minprice = request.GET.get("minprice")
            if flower_minprice:
                flower_minprice = float(flower_minprice)
            flowers_results = []
            for flower in flowers:
                leave = True
                if flower.basesupply.leftnumber > 0:
                    price = flower.basesupply.fixedprice
                    if price == 0:
                        price = flower.basesupply.plannedprice
                        if price == 0:
                            price = flower.basesupply.forecastprice
* (1 + flower.basesupply.addpercentprice) +
flower.basesupply.addprice
                    if flower_maxprice and price > flower_maxprice:
                        leave = False
                    if flower_minprice and price < flower_minprice:
                        leave = False
                    flower.detprice = price
                else:
                    flower_supplies =
Flowersortvariantsupply.objects.filter(flowersortvariant =
flower, supplydate__gt = flower.basesupply.supplydate,
leftnumber__gt = 0)
                    flower_supplies =
flower_supplies.order_by("supplydate")
                    if not flower_supplies:
                        leave = False
                    flower_supply = flower_supplies.first()
                    price = flower_supply.fixedprice
                    if price == 0:
                        price = flower_supply.plannedprice
                        if price == 0:
                            price = flower_supply.forecastprice *
(1 + flower_supply.addpercentprice) + flower_supply.addprice
                    flower.detprice = price

```

```

        if flower_maxprice and price > flower_maxprice:
            leave = False
        if flower_minprice and price < flower_minprice:
            leave = False
    if leave:
        flowers_results.append(flower)
    flower_order = request.GET.get("order")
    if flower_order:
        flower_order = int(flower_order)
        if flower_order == 1:
            flowers_results = sorted(flowers_results,
key=lambda x: x.detprice, reverse=True)
        elif flower_order == 2:
            flowers_results = sorted(flowers_results,
key=lambda x: x.detprice, reverse=False)
        elif flower_order == 3:
            flowers_results = sorted(flowers_results,
key=lambda x: x.score, reverse=True)
        elif flower_order == 4:
            flowers_results = sorted(flowers_results,
key=lambda x: x.score, reverse=False)
        elif flower_order == 5:
            flowers_results = sorted(flowers_results,
key=lambda x: x.height, reverse=True)
        elif flower_order == 6:
            flowers_results = sorted(flowers_results,
key=lambda x: x.height, reverse=False)
        elif flower_order == 7:
            flowers_results = sorted(flowers_results,
key=lambda x: x.flowersort.sortname, reverse=True)
        elif flower_order == 8:
            flowers_results = sorted(flowers_results,
key=lambda x: x.flowersort.sortname, reverse=False)
    flower_interval = request.GET.get("variant")
    if not flower_interval:
        flower_ids = []
        for flowers_result in flowers_results:
            if flowers_result.flowersort not in flower_ids:
                flowersort = flowers_result.flowersort
                flower_ids.append(flowersort)
    else:
        flower_ids = None
    flower_paginator = Paginator(flowers_results, 10)
    flower_interval = request.GET.get("page")
    flower_objects =
flower_paginator.get_page(flower_interval)
    flower_context = {}
    flower_context["flowers"] = flower_objects
    flower_context["sorts"] = flower_ids
    flower_context["searchsorts"] = flower_search_sorts
    kinds = Flowerkind.objects.all()
    sorts = Flowersort.objects.all()
    flower_context["kinds"] = kinds

```

```

        flower_context["allsorts"] = sorts
        return render(request, "flowerspl/flowersearch.html",
flower_context)

def search_bouquet_view(request):
    if not request.user.is_authenticated:
        bouquet_context = {}
        bouquet_context["bouquets"] = None
        return render(request,
"flowerspl/bouquetsearch.html", bouquet_context)
    else:
        flower_user = request.user
        if not flower_user.is_staff:
            bouquet_context = {}
            bouquet_context["bouquets"] = None
            return render(request,
"flowerspl/bouquetsearch.html", bouquet_context)
        bouquets =
Bouquet.objects.select_related("basephoto").all()
        bouquet_caption = request.GET.get("name")
        if bouquet_caption:
            bouquets = bouquets.filter(caption__icontains =
bouquet_caption)
        bouquet_id = request.GET.get("id")
        if bouquet_id:
            bouquet_id = int(bouquet_id)
            bouquets = bouquets.filter(id = bouquet_id)
        flower_kinds = request.GET.get("kinds")
        if flower_kinds:
            flower_kinds_to_search = []
            flower_kinds_separated = flower_kinds.split("@")
            for flower_kind in flower_kinds_separated:
                flower_kind = int(flower_kind)
                flower_kind = Flowerkind.objects.filter(id =
flower_kind)
                flower_kind = flower_kind.first()
                flower_kinds_to_search.append(flower_kind)
            bouque_flowers =
BouquetFlower.objects.select_related("bouquet").filter(flowersor
tvariant__flowerkind__in = flower_kinds_to_search)
            to_bouques = bouque_flowers.values("bouquet__id")
            bouquets = bouquets.filter(id__in = to_bouques)
        flower_sorts = request.GET.get("sorts")
        if flower_sorts:
            flower_sorts_to_search = []
            flower_sorts_separated = flower_sorts.split("@")
            for flower_sort in flower_sorts_separated:
                flower_sort = int(flower_sort)
                flower_sort = Flowersort.objects.filter(id =
flower_sort)
                flower_sort = flower_sort.first()
                flower_sorts_to_search.append(flower_sort)

```

```

        bouquet_flowers =
BouquetFlower.objects.select_related("bouquet").filter(flowersor
tvariant__in = flower_sorts_to_search)
        to_bouques = bouquet_flowers.values("bouquet__id")
        bouquets = bouquets.filter(id__in = to_bouques)
        flower_sizes = request.GET.get("sizes")
        if flower_sizes:
            flower_sizes_to_search = []
            flower_sizes_separated = flower_sizes.split("@")
            for flower_size in flower_sizes_separated:
                flower_size = int(flower_size)
                flower_sizes_to_search.append(flower_size)
            flowers = Flowersortvariant.objects.filter(size__in
= flower_sizes_to_search)
            bouquet_flowers =
BouquetFlower.objects.select_related("bouquet").filter(flowersor
tvariant__in = flowers)
            to_bouques = bouquet_flowers.values("bouquet__id")
            bouquets = bouquets.filter(id__in = to_bouques)
            flower_colors = request.GET.get("colors")
            if flower_colors:
                flower_colors_to_search = []
                flower_colors_separated = flower_colors.split("@")
                for flower_color in flower_colors_separated:
                    flower_color = int(flower_color)
                    flower_colors_to_search.append(flower_color)
                flowers =
Flowersortvariant.objects.filter(color__in =
flower_colors_to_search)
                bouquet_flowers =
BouquetFlower.objects.select_related("bouquet").filter(flowersor
tvariant__in = flowers)
                to_bouques = bouquet_flowers.values("bouquet__id")
                bouquets = bouquets.filter(id__in = to_bouques)
                bouquet_purposes = request.GET.get("purposes")
                if bouquet_purposes:
                    bouquet_purposes_to_search = []
                    bouquet_purposes_separated =
bouquet_purposes.split("@")
                    for bouquet_purposein bouquet_purposes_separated:
                        bouquet_purpose = int(bouquet_purpose)

bouquet_purposes_to_search.append(bouquet_purpose)
                    bouquet_flowers =
BouquetPurpose.objects.select_related("bouquet",
"purpose").filter(purpose__id__in = bouquet_purposes_to_search)
                    to_bouques = bouquet_flowers.values("bouquet__id")
                    bouquets = bouquets.filter(id__in = to_bouques)
                    flower_maxheight = request.GET.get("maxheight")
                    if flower_maxheight:
                        flower_maxheight = float(flower_maxheight)
                        flowers = Flowersortvariant.objects.filter
(height__lte = flower_maxheight)

```

```

        bouquet_flowers =
BouquetFlower.objects.select_related("bouquet").filter(flowersor
tvariant__in = flowers)
        to_bouques = bouquet_flowers.values("bouquet__id")
        bouquets = bouquets.filter(id__in = to_bouques)
        flower_minheight = request.GET.get("minheight")
        if flower_minheight:
            flower_minheight = float(flower_minheight)
            flowers =
Flowersortvariant.objects.filter(height__gte = flower_maxheight)
            bouquet_flowers =
BouquetFlower.objects.select_related("bouquet").filter(flowersor
tvariant__in = flowers)
            to_bouques = bouquet_flowers.values("bouquet__id")
            bouquets = bouquets.filter(id__in = to_bouques)
            bouquet_maxscore = request.GET.get("maxscore")
            if bouquet_maxscore:
                bouquet_maxscore = float(bouquet_maxscore)
                bouquets = bouquets.filter(score__lte =
bouquet_maxscore)
            bouquet_minscore = request.GET.get("minscore")
            if bouquet_minscore:
                bouquet_minscore = float(bouquet_minscore)
                bouquets = bouquets.filter(score__gte =
bouquet_minscore)
            bouquet_maxprice = request.GET.get("maxprice")
            if bouquet_maxprice:
                bouquet_maxprice = float(bouquet_maxprice)
                bouquets = bouquets.filter(Q(fixedprice__lte =
bouquet_maxprice) | Q(currentfixedprice__gt = 0,
currentfixedprice__lte = bouquet_maxprice))
            bouquet_minprice = request.GET.get("minprice")
            if bouquet_minprice:
                bouquet_minprice = float(bouquet_minprice)
                bouquets = bouquets.filter(Q(fixedprice__gte =
bouquet_minprice) | Q(currentfixedprice__gt = 0,
currentfixedprice__gte = bouquet_minprice))
            bouquet_results = []
            bouquet_order = request.GET.get("order")
            if bouquet_order:
                bouquet_order = int(bouquet_order)
                if bouquet_order == 1:
                    bouquet_results = bouquets.order_by("-
fixedprice")
                elif bouquet_order == 2:
                    bouquet_results =
bouquets.order_by("fixedprice")
                elif bouquet_order == 3:
                    bouquet_results = bouquets.order_by("-score")
                elif bouquet_order == 4:
                    bouquet_results = bouquets.order_by("score")
                elif bouquet_order == 5:
                    bouquet_results = bouquets.order_by("caption")

```

```

        elif bouquet_order == 6:
            bouquet_results = bouquets.order_by("-caption")
            bouquet_paginator = Paginator(bouquet_results, 10)
            bouquet_interval = request.GET.get("page")
            bouquet_objects =
bouquet_paginator.get_page(bouquet_interval)
            kinds = Flowerkind.objects.all()
            sorts = Flowersort.objects.all()
            bouquet_context = {}
            bouquet_context["kinds"] = kinds
            bouquet_context["sorts"] = sorts
            return render(request, "flowerspl/bouquetsearch.html",
bouquet_context)

def flowershop_orders_view(request):
    if not request.user.is_authenticated:
        flower_context = {}
        flower_context["objects"] = None
        return render(request,
"flowerspl/flowerorders.html", flower_context)
    else:
        flower_user = request.user
        if not flower_user.is_staff:
            flower_context = {}
            flower_context["bouquets"] = None
            return render(request,
"flowerspl/flowerorders.html", flower_context)
        flowershop_order_status = request.GET.get("status")
        flowershop_orders = FlowershopOrder.objects.all()
        if flowershop_order_status:
            flowershop_order_status =
int(flowershop_order_status)
            if flowershop_order_status != -1:
                flowershop_orders =
flowershop_orders.filter(currentstatus =
flowershop_order_status)
                flowershop_order_startdate =
request.GET.get("start")
                flowershop_order_enddate = request.GET.get("end")
                if flowershop_order_startdate:
                    flowershop_order_startdate =
datetime.strptime(flowershop_order_startdate, '%d-%m-%y %H:%M')
                    flowershop_orders =
flowershop_orders.filter(createdat__gte =
flowershop_order_startdate)
                    if flowershop_order_enddate:
                        flowershop_order_enddate =
datetime.strptime(flowershop_order_enddate, '%d-%m-%y %H:%M')
                        flowershop_orders =
flowershop_orders.filter(createdat__lte =
flowershop_order_enddate)
                        flowershop_order_startdate =
request.GET.get("startdeliv")

```

```

        flowershop_order_enddate =
request.GET.get("enddeliv")
        if flowershop_order_startdate:
            flowershop_order_startdate =
datetime.strptime(flowershop_order_startdate, '%d-%m-%y %H:%M')
            flowershop_orders =
flowershop_orders.filter(deliveryat__gte =
flowershop_order_startdate)
            if flowershop_order_enddate:
                flowershop_order_enddate =
datetime.strptime(flowershop_order_enddate, '%d-%m-%y %H:%M')
                flowershop_orders =
flowershop_orders.filter(deliveryat__lte =
flowershop_order_enddate)
            flowershop_orders = flowershop_orders.order_by("-
createdat")
        flowershop_paginator = Paginator(flowershop_orders,
10)

        flowershop_interval = request.GET.get("page")
        flowershop_objects =
flowershop_paginator.get_page(flowershop_interval)
        flower_context = {}
        flower_context["objects"] = flowershop_objects
        return render(request,
"flowerspl/flowerorders.html", flower_context)

```

ДОДАТОК В
Слайди презентації

Міністерство освіти і науки України
Національний університет «Запорізька політехніка»
кафедра програмних засобів

Тема дипломної кваліфікаційної роботи магістра
«Дослідження та програмна реалізація методів
прогнозування під час постачання живих квітів.
Research and Software Implementation of Methods for
Forecasting the Fresh Flowers Supply»

Виконавець:

Денис МЕРТВІЩЕВ

Керівник: к.т.н., доцент

Валерій ЛЬОВКІН

2024

Рисунок В.1 – Початковий слайд

Об'єкт, предмет, мета роботи

Об'єкт роботи – процеси постачання і продажу живих квітів.

Предмет роботи – методи прогнозування під час постачання живих квітів.

Мета роботи – розробити програмне забезпечення продажу живих квітів для підвищення зручності роботи користувачів на основі інтеграції прогнозування вартості постачання квітів.

Рисунок В.2 – Об'єкт, предмет, мета роботи

Завдання роботи

- проаналізувати програми продажу живих квітів;
- розробити метод прогнозування вартості постачання живих квітів;
- виконати проектування програми продажу живих квітів;
- реалізувати програму продажу живих квітів;
- виконати експериментальне дослідження прогнозування вартості постачання живих квітів.

3

Рисунок В.3 – Завдання роботи

Порівняння аналогів розробки вебзастосунку продажу живих квітів

Показник	Flowers.ua	Dicentra	Розробка
Перегляд повної інформації про букет	+	+	+
Оцінювання або відгуки про букети живих квітів	+	+	+
Внесення букетів живих квітів до бажань	–	–	+
Багатокритеріальний підбір або пошук живих квітів	–	+	+
Замовлення букетів живих квітів	+	+	+
Замовлення живих квітів	–	–	+
Інтеграція засобів прогнозування вартості живих квітів	–	–	+
Визначення постачань живих квітів	–	–	+

4

Рисунок В.4 – Порівняння аналогів розробки вебзастосунку продажу живих квітів

Метод прогнозування вартості постачання живих квітів

Підготовка даних у вигляді навчальної вибірки даних, включаючи ряд значень ознак для кожного екземпляру даних. Кожен екземпляр даних представляє проведений аукціон з продажу живих квітів. Усі екземпляри даних стосуються аукціонів з продажу квітів певного сорту. Після того, як для кожного екземпляру даних визначено набір значень за кожної з відповідних ознак, до екземпляру має бути додано вартість постачання живих квітів, яка була отримана в даному аукціоні. Усі значення вартостей мають бути пронормовані, а тоді пронормовані значення мають бути додані до відповідних екземплярів.

На наступному етапі формується модель для прогнозування вартості постачання живих квітів на основі створення ансамблю дерев рішень за допомогою методу Random Forest. Кожне дерево спеціалізується на певній частині навчальної вибірки даних, а підсумкове рішення про вартість постачання приймається шляхом усереднення значень, отриманих прогнозуванням вартості кожним з дерев ансамблю.

Сформована таким чином модель застосовується для навчання на основі створеної на першому етапі вибірки даних.

Останній етап реалізується при появі кожного нового екземпляру даних, для якого невідома вартість постачання живих квітів, якщо ці живі квіти належать до заданого сорту, що продається на цих аукціонах. Тоді такий аукціон має бути визначений на основі значень вхідних ознак, формуючи таким чином екземпляр даних, після чого він подається на вхід навченої моделі, а на виході отримується результат прогнозування — вартість постачання одиниці живих квітів.

5

Рисунок В.5 – Метод прогнозування вартості постачання живих квітів

Вибір засобів розробки програми продажу живих квітів

	Wordpress	Django
Тип забезпеченого рішення	Система управління вмістом	Фреймворк веброзробки
Основна мова розробки	PHP	Python
Простота створення стандартних функцій магазину	Вища	Нижча
Простота створення додаткових функцій магазину	Нижча	Вища
Безпечність підсумкового рішення	Нижча	Висока
Рівень підтримки для створення моделей прогнозування відповідною мовою розробки	Нижчий	Високий

6

Рисунок В.6 – Вибір засобів розробки програми продажу живих квітів

Діаграми варіантів використання програми магазину продажу живих квітів

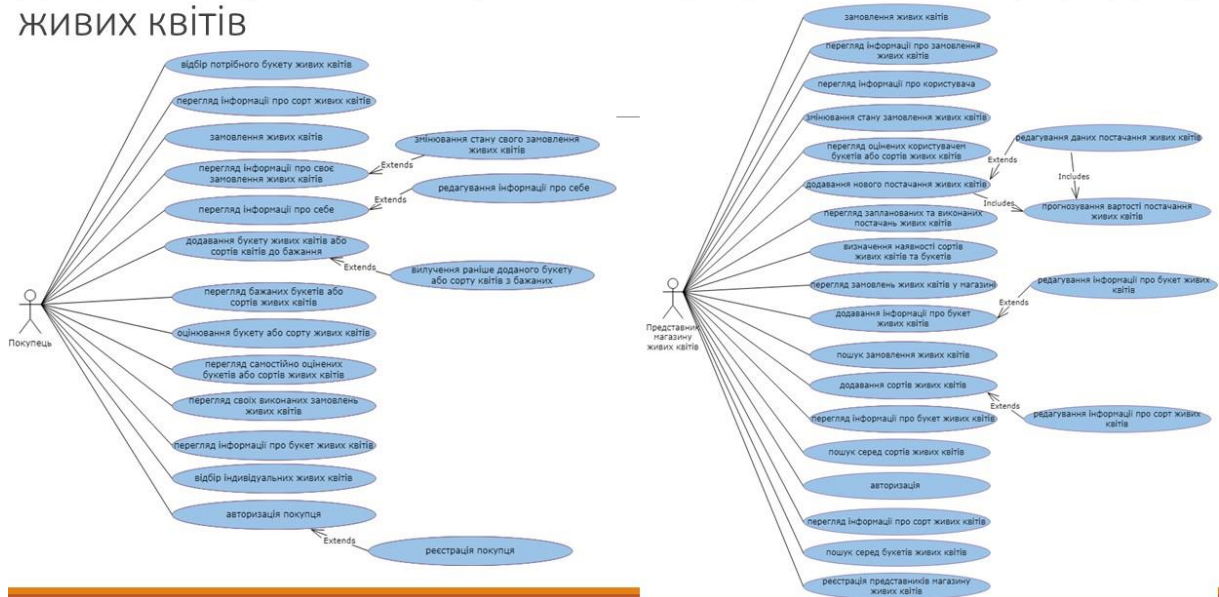


Рисунок В.7 – Діаграми варіантів використання програми магазину продажу живих квітів

Схема бази даних

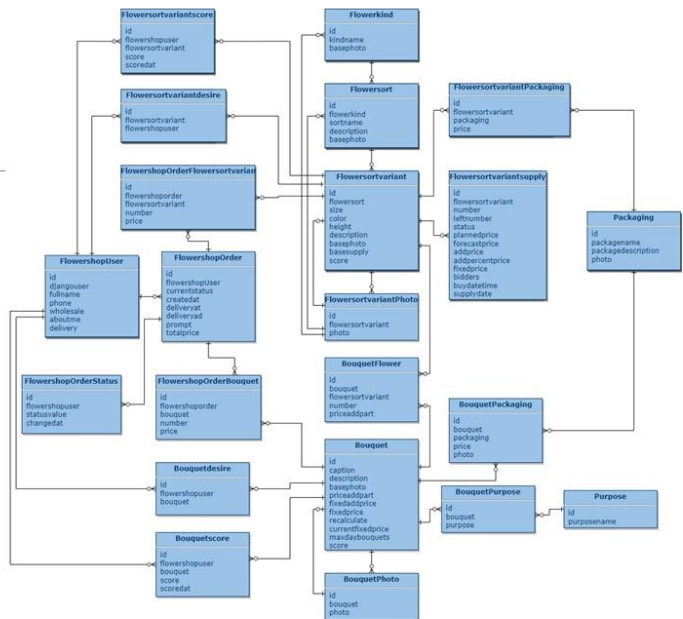


Рисунок В.8 – Схема бази даних

Діаграми станів

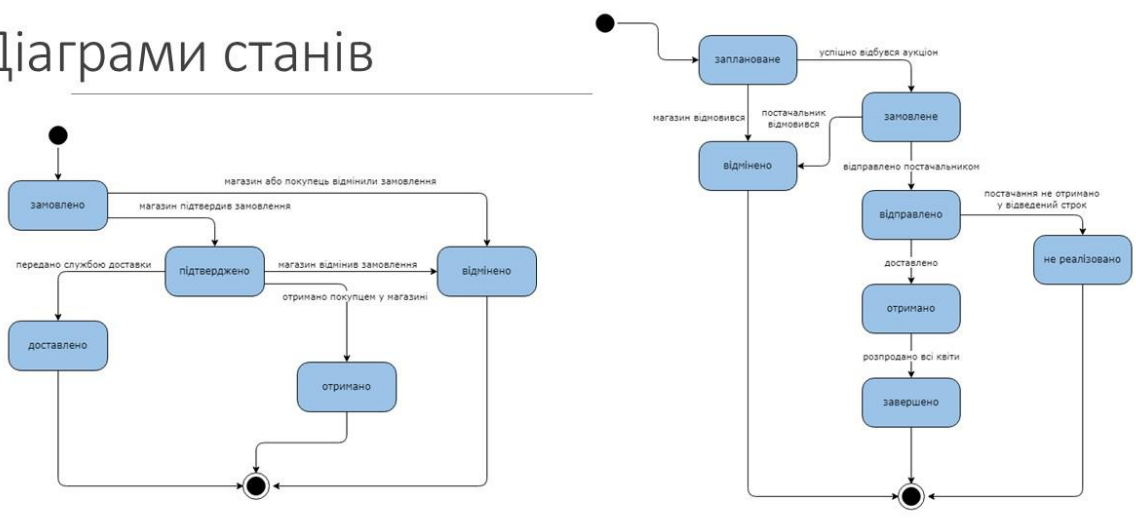


Рисунок В.9 – Діаграми станів

Відтворення власних бажань авторизованого користувача у програмі

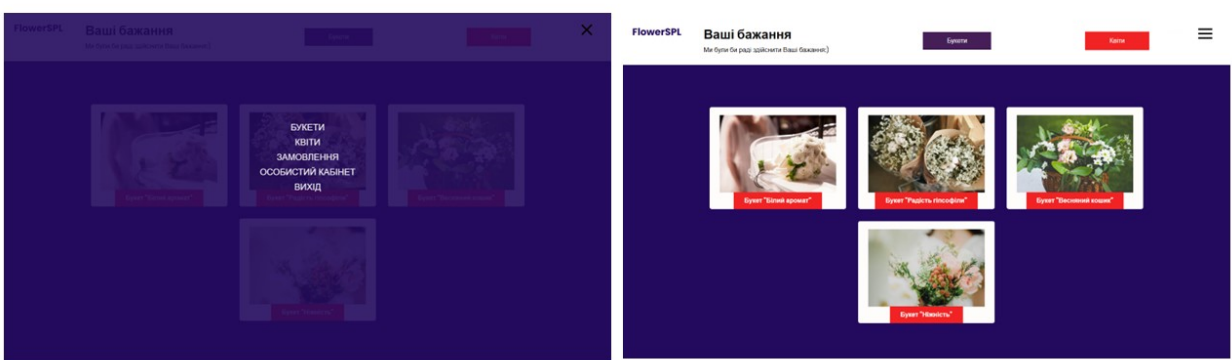
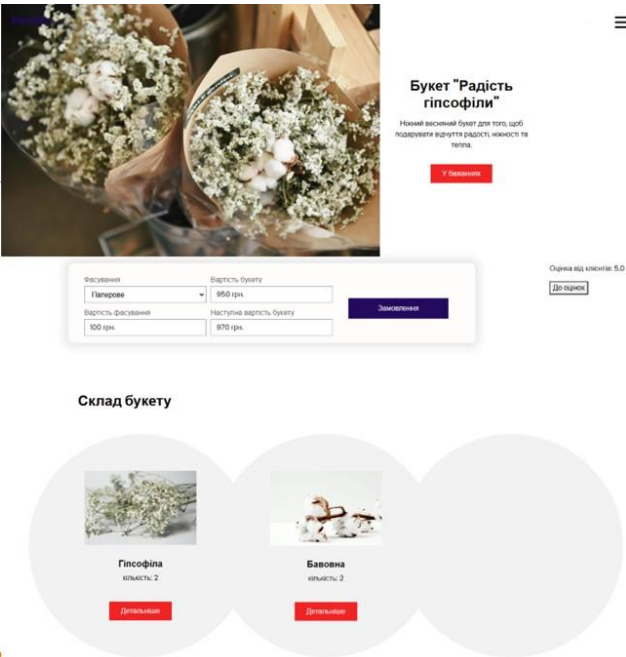


Рисунок В.10 – Відтворення власних бажань авторизованого користувача у програмі

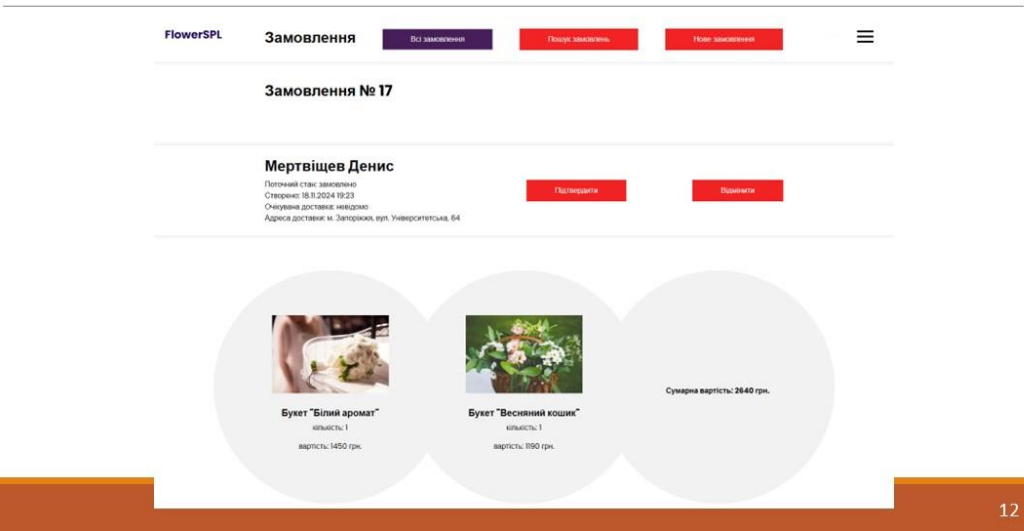
Сторінка букету живих квітів у програмі



11

Рисунок В.11 – Сторінка букету живих квітів у програмі

Сторінка замовлень у магазині продажу живих квітів



12

Рисунок В.12 – Сторінка замовлень у магазині продажу живих квітів

Результати прогнозування вартості постачання живих квітів

Кількість дерев у ансамблі	MAE	MSE	RMSE
Random Forest			
50	0,0246055	0,0010694	0,0327014
100	0,0246873	0,0010159	0,0318732
300	0,0238526	0,0010118	0,0318085
500	0,0234413	0,0009583	0,0309558
600	0,0239213	0,0010009	0,0316373
700	0,0242298	0,00101	0,0317807
750	0,0236311	0,0009766	0,0312502
900	0,0235497	0,0009705	0,0311536
1000	0,0238729	0,001003	0,0316706

Спосіб створення моделі	MAE	MSE	RMSE
Метод опорних векторів	0,0453399	0,0027406	0,0523512
CART	0,0262265	0,0016424	0,040526
Запропонований метод прогнозування вартості постачання живих квітів	0,0234413	0,0009583	0,0309558

13

Рисунок В.13 – Результати прогнозування вартості постачання живих квітів

Висновки

За підсумками виконання роботи розроблено програмне забезпечення продажу живих квітів для підвищення зручності роботи користувачів на основі інтеграції прогнозування вартості постачання квітів.

Створений вебзастосунок продажу живих квітів дозволяє виконувати прогнозування вартості постачання живих квітів, враховуючи це при формуванні очікуваної вартості продажу квітів для гуртових та індивідуальних покупців, формувати замовлення букетів та сортів квітів, надає засоби для підтримки знаходження потрібних квітів та букетів, дозволяє висловлювати власні оцінки стосовно придбаних живих квітів та букетів.

Наукова новизна роботи полягає в методі прогнозування вартості постачання живих квітів, який визначає створення і навчання моделі ансамблю дерев рішень на основі методу Random Forest, що дозволяє визначити вартість продажу одиниці квітів на аукціоні. Експериментальне дослідження підтвердило зменшення похибки прогнозування вартості живих квітів порівняно з методом опорних векторів та окремим деревом рішень на основі алгоритму CART.

14

Рисунок В.14 – Висновки