

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ЗАМОВЛЕННЯ ЇЖИ ДОМАШНЬОГО
ПРИГОТУВАННЯ
HOMEMADE FOOD ORDERING SOFTWARE

Виконав: студент(ка) 4 курсу, групи КНТ-138

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Тарусов В.О.

(прізвище та ініціали)

Керівник Голуб Т.В.

(прізвище та ініціали)

Рецензент Терещенко Е.В.

(прізвище та ініціали)

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

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

ЗАТВЕРДЖУЮ

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

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Тарусова Владислава Олександровича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення замовлення їжі домашнього приготування
Homemade Food Ordering Software

керівник проєкту (роботи) Голуб Тетяна Василівна, к.т.н.,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

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

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

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

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

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

6. Консультанти розділів проєкту (роботи)

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

7. Дата видачі завдання «11» квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка)

(підпис)

Тарусов В.О.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Голуб Т.В.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 84 с., 22 рис., 2 табл., 3 дод., 11 джерел.

СТРАВА, ДОМАШНЄ ПРИГОТУВАННЯ, КУХАР, БАЗА ДАНИХ, ВЕБЗАСТОСУНОК.

Об'єктом роботи є процес замовлення їжі домашнього приготування. Предметом роботи є програмне забезпечення замовлення їжі домашнього приготування. Мета роботи – розробити програмне забезпечення замовлення їжі домашнього приготування для встановлення взаємодії між кухарями і клієнтами.

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

Результати. Приготування їжі вдома розглянуто як засіб для зароблення коштів в умовах скорочення заробітку, бажання реалізовуватися у такий спосіб. Базовою сутністю виділено замовлення, для якого визначено стани, що лягли в основу проєктування програми, включаючи структури даних, структуру сторінок, представлень програми, порядок взаємодії з програмою.

Висновки. Розроблена програма дозволяє підтримувати роботу бажаючих готувати вдома для інших людей шляхом організації замовлень таких страв, керування цими замовленнями, підтримувати клієнтів, які бажають визначити те, які страви прийдуть їм до смаку і знайти серед цього різноманіття саме те, що дозволить харчуватися домашньою і до того ж можливо здоровою їжею.

Галузь використання. Готування їжі вдома, організація підтримки роботи з замовленнями для цієї сфери.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 84 pages, 22 figures, 2 tables, 3 appendixes, 11 sources.

DISH, HOMEMADE COOKING, COOK, DATABASE, WEBSITE.

The object of the work is the process of providing homemade food. The subject of the work is homemade food software. The aim of the work is to develop homemade food software to establish interaction between cooks and customers.

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

Results. Cooking at home is considered as a source for earning money in the face of wage cuts, the desire to be realized in this way. The basic concept of the subject area is an order which structured the software, including data structures, webpage structure, organizing program views, order of interaction with the program.

Conclusions. The developed program provides support to those who want to work at home for other people by organizing such dishes, managing these orders, supporting customers who want to choose what dishes are accepted and find among the variety exactly such food which allows them to eat at home and to eat possibly healthy food.

Field of use. Place of residence, support of work with orders for this sphere.

ЗМІСТ

С.

Перелік скорочень та умовних познач	7
Вступ	8
1 Аналіз предметної області	9
1.1 Особливості харчування їжею домашнього приготування	9
1.2 Аналіз аналогів	10
1.3 Висновки за розділом 1	12
2 Розробка архітектури програми	14
2.1 Вибір засобів розробки програми	14
2.2 Функціональні вимоги до програмного забезпечення	14
2.3 Проектування бази даних	17
2.4 Висновки за розділом 2	27
3 Розробка програми	28
3.1 Робота з замовленнями їжі домашнього приготування	28
3.2 Опис реалізації програми	29
3.3 Висновки за розділом 3	41
4 Експлуатація та тестування програми	42
4.1 Призначення програми	42
4.2 Умови виконання програми	42
4.3 Тестування і опис принципів експлуатації програми	42
4.4 Висновки за розділом 4	48
Висновки	49
Перелік джерел посилання	50
Додаток А Технічне завдання	52
Додаток Б Текст програми	56
Додаток В Слайди презентації	78

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;

ПЗ – програмне забезпечення.

ВСТУП

Ресторанний бізнес у розвинутих країнах дуже активно увійшов у життя громадян. Але домашнє харчування має свої особливості, дозволяє забезпечувати здорове харчування, дозволяє контролювати харчування, проте потребує і додаткових зусиль і часу.

Зараз для багатьох людей приготування їжі самостійно є проблемою, проте вони можуть бажати все одно харчуватися домашньою їжею. З іншого боку інші люди готові витратити час на приготування страв і заробляти за це гроші. До того ж це додатково стає важливим в кризовий період, в умовах війни, коли підтримка малих бізнесів стає критичною. Саме для створення контакту між людьми, готовими готувати вдома страви, і їх потенційними клієнтами і має використовуватися розроблювана програма.

Додатковий вплив, що визначає актуальність роботи, визначила пандемія коронавірусу: раптова стресова перевірка, якій піддалися багато ланцюгів поставок на початку карантину, означала, що деякий час готових страв у споживачів не вистачало, а їсти в місцевому ресторані перестало бути можливістю, однак згідно з опитуванням людей, які проживають у п'яти європейських економіках, навіть після завершення пандемії переважна більшість споживачів буде покладатися на домашню їжу на вечерю або більшу кількість разів на день [1]-[2].

Грунтуючись на визначеній актуальності приготування домашньої їжі загалом та наявності ніші для замовлення цієї їжі у інших людей, було визначено мету роботи – розробити програмне забезпечення (ПЗ) замовлення їжі домашнього приготування для встановлення взаємодії між кухарями і клієнтами.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Особливості харчування їжею домашнього приготування

Харчування їжею домашнього приготування має наступні переваги:

- економія грошей: їсти домашню їжу зазвичай набагато дешевше, ніж їсти в ресторані або купувати оброблені продукти на ринку, адже під час харчування в ресторані оплачується не лише їжа, а й витрати на ведення цього бізнесу (світло, вода, оренда будівлі та персонал) на додаток до їжі, яку їмо, звичайно, що якщо замовляти домашню їжу, то це не буде на стільки дешево, як готувати самим, проте це все одно має бути дешевше ніж їсти таку саму саме страву в ресторані;

- економія часу: може здатися, що швидким рішенням, коли людина поспішає, може бути придбати щось перекусити в місцевому супермаркеті або поїхати взяти їжу на винос до найближчого ресторану, але насправді багато разів приготувати щось вдома може бути набагато швидше, особливо якщо все планується заздалегідь, окрім того можна приготувати одразу більше, а з'їсти надлишок пізніше протягом тижня або заморозити його, так само можна поступити і зі стравами, замовленими в кухарів, які їх приготують у себе вдома;

- більш здорові інгредієнти: багато комерційно приготованих продуктів містять багато жиру, солі та цукру, коли людина готує власну їжу, вона точно знає, які інгредієнти і скільки кожного з них потрапляє в їжу, так само можна вибирати екологічні або здорові варіанти, якщо замовляти їжу домашнього приготування;

- уникнення харчової алергії та чутливості: приготування їжі вдома може бути особливо корисним, якщо у людини є харчова алергія, оскільки ви контролюється власна кухня, продукти, то можна знизити ризик алергічної реакції;

- зближення сім'ї: їжа вдома дає можливість всій родині поговорити про свій день [3].

1.2 Аналіз аналогів

Застосунок HomeMade (рис. 1.1) [4]-[5] допомагає знайти кухарів, які готують в себе вдома домашню їжу поблизу з собою, та доручити їм готувати, для цього потрібно переглянути меню, які вони пропонують, і вибрати те, яке клієнту найбільше подобається, після цього потрібно вибрати страви, які клієнт хотів би з'їсти, і дочекатися їх доставки прямо до дверей, можна при цьому відстежувати своє замовлення, крім того можна також замовити самовивіз їжі, особливістю є те, що застосунок показує лише кухарів у районі клієнта, щоб переконатися, що клієнт отримає свіжоприготовані гарячі страви, готові до вживання, можна також шукати конкретні страви або встановлювати певні фільтри за власним бажанням [6]-[7].

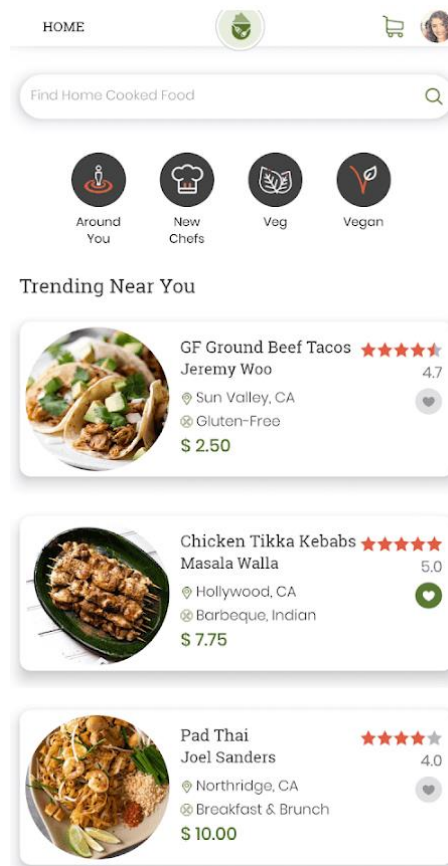


Рисунок 1.1 – Програма HomeMade [5]

Savorly (рис. 1.2) [8]-[9] є ще одним подібним рішенням: користувач також переглядає перелік домашніх кухарів у своєму районі та вибирає страви з тих, які йому подобаються, при цьому додаток також дозволяє здійснювати пошук за допомогою фільтрів страв, є такі фільтри, як вегетаріанський, веганський, органічний тощо, крім того, можна переглянути всі доступні страви за категоріями, наприклад, сніданок, вечеря тощо, усі страви мають детальний список інгредієнтів, фотографії, рейтинг та всю іншу інформацію, також можна бачити відстань між клієнтом та кухарем, у якого він збирається зробити замовлення, також можете поспілкуватися з кухарем для особливих запитів або запитань, проте на даний момент програмою не підтримується робота зі службами доставки [6]-[7].

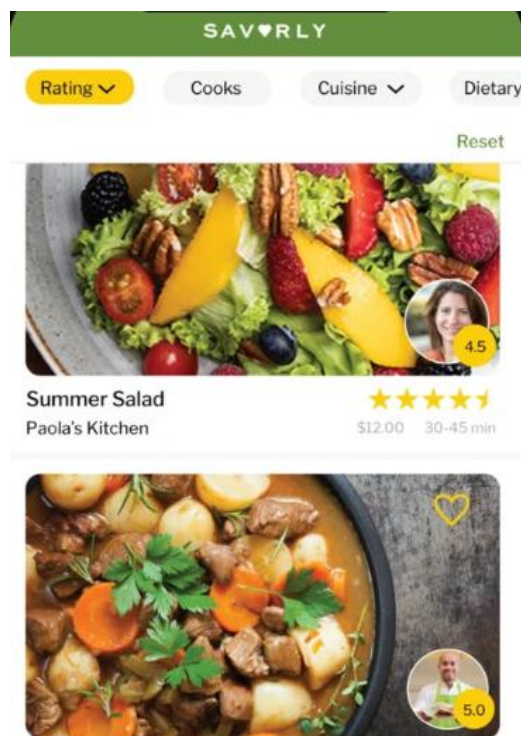


Рисунок 1.2 – Програма Savorly [9]

У табл. 1.1 виконано порівняння існуючих засобів і розробки, яка має бути виконана в роботі. Основною проблемою програм-аналогів є те, що всі вони недоступні для роботи в Україні і фактично покривають територію Сполучених Штатів Америки, що є проблемою для використання. Окрім того

Savorly не працює зі службами доставки. У моїй програмі додатково до цих відмінностей заплановано надати клієнту можливості швидкого доступу до сторінок певних кухарів, а також розширення фільтрів страв, за якими страви обираються.

Таблиця 1.1 – Порівняння існуючих засобів і розробки

Критерій	HomeMade	Savorly	Моя програма
Доступність в Україні	Ні	Ні	Так
Пошук страв домашнього приготування	Так	Так	Так
Визначення фільтрів страв	Обмежене	Обмежене	Розширюване
Замовлення страв	Так	Так	Так
Перелік продуктів у страві	Так	Так	Так
Швидкий доступ до обраних страв	Так	Так	Так
Швидкий доступ до обраних кухарів	Ні	Ні	Так
Вибір служби доставки	Так	Ні	Так

1.3 Висновки за розділом 1

Виконано огляд особливостей, на які дозволяє сподіватися вживання домашньої їжі. Окрім смакових особливостей така їжа також є більш здоровою, дозволяє контролювати використання різних домішок, продуктів, все це може бути дуже важливим, окрім того це можливість харчуватися у себе вдома. Проведено огляд аналогів, включаючи застосунки HomeMade і Savorly. Визначено особливості власної розробки, яка на відміну від них повинна мати розширюваний набір фільтрів щодо продуктів, які використовуються при приготуванні страви, можливість швидкого доступу

до сторінок кухарів.

Для виконання розробки програми було встановлено наступні завдання в роботі:

- дослідження проблеми;
- визначення вимог до розробки;
- проєктування ПЗ замовлення їжі домашнього приготування;
- програмна реалізація ПЗ замовлення їжі домашнього приготування.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір засобів розробки програми

Веброзробка дуже часто реалізується мовами Java і Python [10], які і були альтернативами вибору мови програмування (табл. 2.1). Остаточного було обрано Python, враховуючи кращу читабельність його коду, що сприяє кращому сприйняттю написаного під час подальшої розробки та під час вивчення нових можливостей кращому сприйняттю прикладів, до того ж ця мова характеризується вищою динамічністю розробки, що пов'язано з тим, що код зазвичай є коротким, точним і швидким для вивчення.

Таблиця 2.1 – Вибір мови розробки програми

Критерій	Python	Java
Динамічність розробки	Вища	Нижча
Читабельність коду	Вища	Нижча
Фреймворки	Django, Flask	Spring, Blade
Стабільність	Однакова	Однакова

Вибір Python продовжено вибором Django [11] як вебфреймворку.

2.2 Функціональні вимоги до програмного забезпечення

Програма має надавати можливість користувачам працювати з такими функціями:

- визначення страв, які готує кухар;
- створення відгуків про страви у кухаря;
- створення відгуків про кухаря;
- замовлення домашніх страв у кухаря;
- керування замовленнями страв;

- пошук домашніх страв;
- перегляд замовлень клієнтом;
- перегляд відгуків про кухаря;
- збереження страв кухаря у власний перелік для швидкого доступу;
- збереження даних кухаря у власний перелік для швидкого доступу;
- надсилання повідомлень;
- відтворення переписки;
- перегляд страв кухаря;
- авторизація користувачів;
- реєстрація клієнтів і кухарів.

Сценарії роботи клієнта з програмою (рис. 2.1):

- створення відгуків про страви у кухаря;
- створення відгуків про кухаря;
- замовлення домашніх страв у кухаря;
- пошук домашніх страв;
- перегляд власних замовлень;
- перегляд відгуків про кухаря;
- робота з панеллю швидкого доступу;
- робота над перепискою з кухарями;
- перегляд страв кухаря;
- авторизація.

Альтернативними сценаріями до них є:

- керування власними замовленнями страв під час перегляду власних замовлень, якщо замовлення потрібно змінити через неточності;
- реєстрація клієнта під час авторизації, якщо клієнт не має облікового запису.

Сценарії роботи кухаря з програмою (рис. 2.2):

- визначення страв, які готує кухар;
- керування замовленнями страв;
- перегляд відгуків;

- робота над перепискою з клієнтами;
- авторизація.

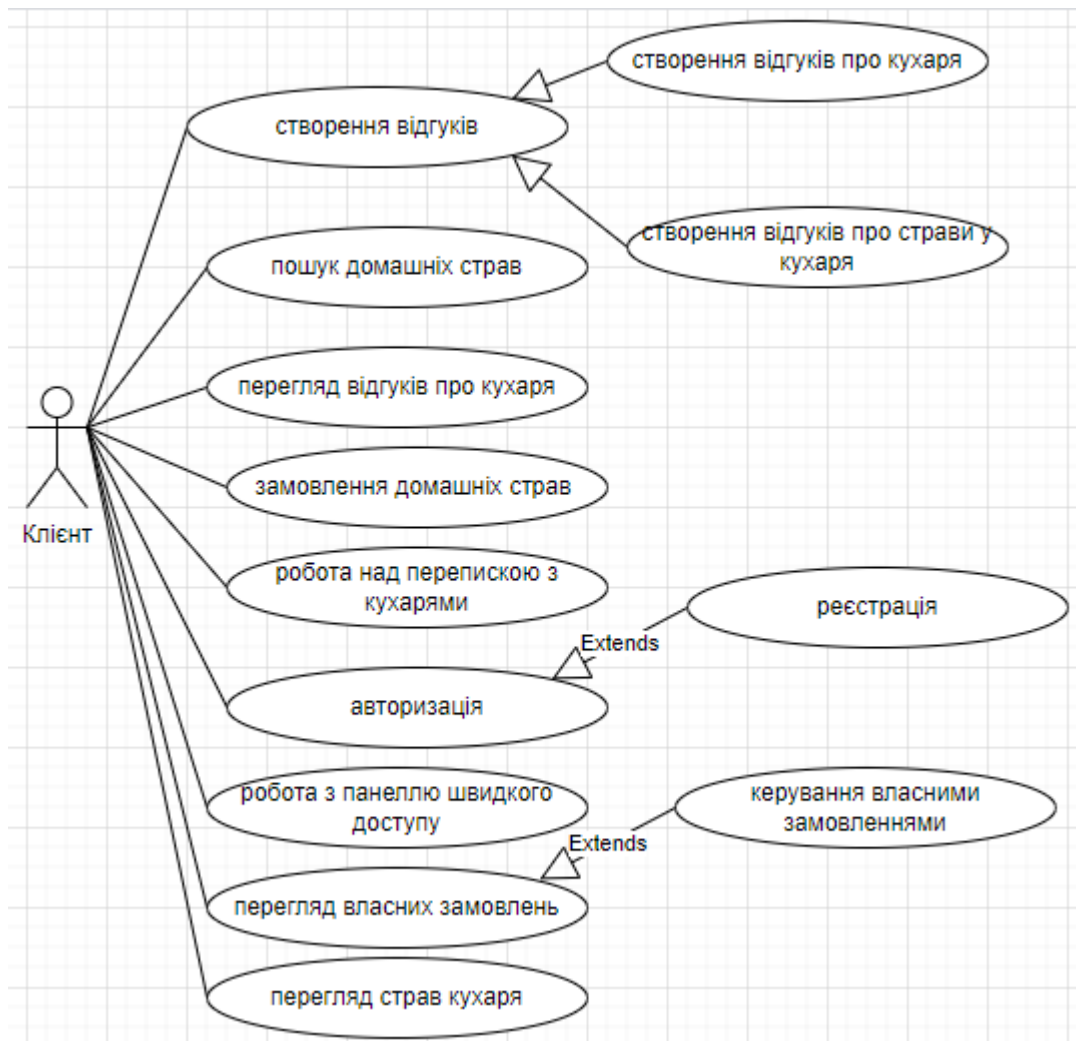


Рисунок 2.1 – Діаграма варіантів використання системи клієнтом



Рисунок 2.2 – Діаграма варіантів використання системи кухарем

На цьому рівні альтернативним сценарієм був виділений тільки сценарій реєстрації, який реалізується у випадку відсутності облікового запису у кухаря під час спроби авторизації.

2.3 Проєктування бази даних

До сутностей бази даних (БД), виділених у процесі проєктування, віднесено:

- кухар;
- клієнт;
- страва;
- реалізація страви у кухаря;
- категорія страви;
- замовлення;
- склад замовлення;
- продукт;
- категорія продукту;
- позначка;
- вартість страви у виконанні кухаря;
- відгук про страву у виконанні кухаря;
- відгук про кухаря;
- повідомлення між кухарем і клієнтом;
- доставка;
- збереження кухаря клієнтом;
- збереження страви у виконанні певного кухаря клієнтом.

Реалізацію роботи з БД виконано за допомогою системи управління БД PostgreSQL.

Таблиця Cook представляє сутність кухаря, використовуючи такі поля таблиці:

- id – номер кухаря;

- first_name – ім'я кухаря;
- last_name – прізвище кухаря;
- username – ім'я кухаря як користувача в системі;
- password – пароль кухаря;
- address – адреса, де кухар готує їжу;
- rating – рейтинг кухаря;
- since – дата, з якої кухар почав працювати з системою;
- phone – номер телефону кухаря.

Таблиця Client представляє сутність клієнта, використовуючи такі поля таблиці:

- id – номер клієнта;
- first_name – ім'я клієнта;
- last_name – прізвище клієнта;
- username – ім'я клієнта як користувача в системі;
- password – пароль клієнта;
- address – адреса клієнта, яка стандартно використовується для доставки;
- phone – номер телефону клієнта.

Таблиця Food представляє сутність страви, використовуючи такі поля таблиці:

- id – номер страви;
- named – назва страви;
- foodcategoryid – категорія страви, що виражає відповідний номер категорії;
- about – опис страви.

Поле foodcategoryid таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею Foodcategory, визначаючи відношення один до багатьох: одній категорії страв відповідає множина страв у довіднику, а одна страва може відноситись у свою чергу до однієї конкретної категорії.

Таблиця CookFood представляє сутність реалізації страви у кухаря, використовуючи такі поля таблиці:

- id – номер реалізації страви кухарем;
- cookid – кухар, який готує цю страву;
- foodid – страва, яку готує кухар;
- alternativenamed – альтернативна назва страви у цього кухаря або назва страви, якої немає в довіднику страв;
- since – дата, з якої цей кухар готує цю страву, надаючи про це дані в системі;
- active – поле, яке визначає, чи це активний запис, чи ні;
- recipe – особливості приготування страви у цього кухаря;
- rating – рейтинг страви у виконанні цього кухаря.

Поле cookid таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею Cook, визначаючи відношення один до багатьох: один господар може готувати множину власних страв, а одна власна страва кухаря може бути приготовлена тільки одним господарем.

Поле foodid таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею Food, визначаючи відношення один до багатьох: одній stravі може бути співставлена множина власних страв, тобто приготовлених різними кухарями, а одна власна страва може реалізовувати тільки одну страву з довідника.

Таблиця Foodcategory представляє сутність категорії страв, використовуючи такі поля таблиці:

- id – номер категорії страв;
- named – назва категорії страв;
- about – опис категорії страв.

Таблиця Order представляє сутність замовлення, використовуючи такі поля таблиці:

- id – номер замовлення;
- clientid – клієнт, який створив замовлення, представлена її номером;

- created – дата створення замовлення;
- for_date – дата і час, на який повинно бути готове замовлення;
- comment – коментар до замовлення;
- pay – сума, яку необхідно сплатити за замовлення;
- deliveryid – доставка, яка буде використовуватися для доставки замовлення, представлена її номером;
- state – стан замовлення.

Поле clientid таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею Client, визначаючи відношення один до багатьох: одному клієнту відповідає множина його власних замовлень, одне замовлення може бути створено тільки одним клієнтом.

Поле deliveryid таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею Delivery, визначаючи відношення один до багатьох: однією службою доставки або одним способом доставки може доставлятися множина замовлень, але одне замовлення може бути доставлено тільки однією службою доставки або способом, замовлення не може бути розділено при транспортуванні.

Таблиця OrderCookFood представляє склад замовлення, використовуючи такі поля таблиці:

- id – номер складової замовлення;
- orderid – замовлення, до складу якого входить ця страва, представлене номером замовлення;
- cookfoodid – страва авторства певного кухаря, що входить до замовлення, представлена номером відповідного запису;
- comment – коментар до особливостей страви у цьому замовленні;
- amount – кількість одиниць цієї страви у замовленні;
- pay – вартість одиниці страви на момент замовлення.

Поле orderid таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею Order, визначаючи відношення один до багатьох: одне замовлення

може включати множину позицій з різними стравами у виконанні кухаря, але кожна позиція належить тільки одному замовленню.

Поле `cookfoodid` таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею `CookFood`, визначаючи відношення один до багатьох: одна страва у виконанні кухаря може входити до складу позицій різних замовлень, але одна позиція може передбачати тільки одну страву.

Таблиця `Category` представляє категорію продукту, використовуючи такі поля таблиці:

- `id` – номер категорії продукту;
- `named` – назва категорії продукту.

Таблиця `Product` представляє продукт, використовуючи такі поля таблиці:

- `id` – номер продукту;
- `named` – назва продукту;
- `categoryid` – категорія продукту, представлена її номером.

Поле `categoryid` таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею `Category`, визначаючи відношення один до багатьох: одна категорія продукту в довіднику може включати множину різних продуктів, але один продукт може відноситися тільки до однієї категорії.

Таблиця `Mark` представляє позначку, використовуючи такі поля таблиці:

- `id` – номер позначки;
- `named` – назва позначки про продукт.

Таблиця `CookFoodProduct` представляє продукт, з якого виготовляється страва, використовуючи такі поля таблиці:

- `id` – номер;
- `productid` – продукт, що входить до складу страви, представлений номером продукту;
- `cookfoodid` – реалізація страви кухарем, в якій використовується цей продукт, представлена номером цієї реалізації.

Поле `productid` таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею `Product`, визначаючи відношення один до багатьох: один продукт може входити до множини позицій в складі різних страв у виконанні певних кухарів (тобто рецептів), але одна позиція в складі конкретної страви співвідноситься з одним конкретним продуктом.

Поле `cookfoodid` таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею `CookFood`, визначаючи відношення один до багатьох: одна страва кухаря може містити множину різних продуктів, але кожна окрема така позиція відповідає тільки одному продукту.

Таблиця `CookFoodProductMark` представляє позначку, встановлену продукту, з якого виготовляється страва, використовуючи такі поля таблиці:

- `id` – номер;
- `cookfoodproductid` – продукт, з якого виготовляється страва, представлений номером продукту;
- `markid` – позначка, встановлена цьому продукту, представлена номером позначки.

Поле `cookfoodproductid` таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею `CookFoodProduct`, визначаючи відношення один до багатьох: конкретний продукт саме з конкретної страви може мати множину позначок, але кожний запис про таку позначку відповідає тільки одному конкретному продукту.

Поле `markid` таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею `Mark`, визначаючи відношення один до багатьох: конкретна позначка може співвідноситися з множиною конкретних продуктів у конкретних стравах, але кожен такий запис про позначку продукту стосується тільки однієї конкретної позначки.

Таблиця `CookFoodPrice` представляє вартість страви у виконанні кухаря, використовуючи такі поля таблиці:

- `id` – номер;

- `cookfoodid` – страва у виконанні певного кухаря, якій встановлюється вартість;

- `from_date` – дата, з якої встановлена ця вартість страви;

- `to_date` – дата, до якої встановлена ця вартість страви;

- `price` – вартість страви у цей період;

- `about` – особливості страви у цей період.

Поле `cookfoodid` таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею `CookFood`, визначаючи відношення один до багатьох: страва у виконанні конкретного кухаря має множину цінових пропозицій, але кожна цінова пропозиція стосується тільки однієї страви у виконанні кухаря.

Таблиця `CookFoodRespond` представляє відгук про страву у виконанні кухаря, використовуючи такі поля таблиці:

- `id` – номер;

- `cookfoodid` – страва у виконанні певного кухаря, якої стосується відгук, номер запису;

- `clientid` – клієнт, який створив цей відгук, представлений його номером;

- `respond` – відгук;

- `created` – дата створення відгуку;

- `rating` – оцінка у відгуку.

Поле `cookfoodid` таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею `CookFood`, визначаючи відношення один до багатьох: страва у виконанні конкретного кухаря має множину відгуків, але кожен відгук стосується тільки однієї страви.

Поле `clientid` таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею `Client`, визначаючи відношення один до багатьох: клієнт може мати множину відгуків про різні страви у виконанні конкретних кухарів, але кожен такий відгук може бути створений тільки один клієнтом.

Таблиця `CookRespond` представляє відгук про кухаря, використовуючи такі поля таблиці:

- id – номер відгуку;
- cookid – кухар, про якого створено відгук, представлений номером кухар;
- clientid – клієнт, який створив відгук, представлений номером клієнта;
- respond – відгук;
- created – дата, коли було створено відгук про кухаря;
- rating – оцінка у відгуку.

Поле cookid таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею Cook, визначаючи відношення один до багатьох: кожен кухар може мати множину відгуків, але кожен відгук стосується тільки одного кухаря.

Поле clientid таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею Client, визначаючи відношення один до багатьох: клієнт може мати множину відгуків про різних кухарів, але кожен такий відгук може бути створений тільки один клієнтом.

Таблиця SavedCook представляє дані збереженого кухаря, використовуючи такі поля таблиці:

- id – номер;
- clientid – клієнт, який зберіг дані кухаря, представлений номером клієнта;
- cookid – кухар, якого зберіг клієнт, представлений номером кухаря.

Поле cookid таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею Cook, визначаючи відношення один до багатьох: кожен кухар може бути збережений у множини різних клієнтів, але кожне збереження стосується тільки одного кухаря.

Поле clientid таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею Client, визначаючи відношення один до багатьох: клієнт може мати множину збережених кухарів, але кожне таке збереження може бути виконане тільки один клієнтом.

Таблиця SavedCookFood представляє збережену страву у виконанні кухаря, використовуючи такі поля таблиці:

- id – номер;
- clientid – клієнт, який зберіг цю страву, представлений номером клієнта;
- cookfoodid – страва у виконанні певного кухаря, яку зберіг клієнт, представлена номером цієї страви.

Поле clientid таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею Client, визначаючи відношення один до багатьох: клієнт може мати множину збережених страв різних кухарів, але кожне таке збереження може бути виконане тільки один клієнтом.

Поле cookfoodid таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею CookFood, визначаючи відношення один до багатьох: одна страва у виконанні певного кухаря може бути збережена у множини клієнтів, але кожне таке збереження стосується тільки однієї страви у виконанні певного кухаря.

Таблиця Message представляє повідомлення між кухарем і клієнтом, використовуючи такі поля таблиці:

- id – номер повідомлення;
- clientid – клієнт, який створив або якому адресовано повідомлення, представлений номером клієнта;
- cookid – кухар, який створив або якому адресовано повідомлення, представлений номером кухаря;
- direction – напрям повідомлення (адресоване кухарю або клієнту);
- about – текст повідомлення;
- created – дата створення повідомлення.

Поле clientid таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею Client, визначаючи відношення один до багатьох: клієнт може створити або йому адресуватися множина повідомлень, але кожне таке повідомлення може бути створено або стосуватися тільки одного клієнта.

Поле `cookid` таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею `Cook`, визначаючи відношення один до багатьох: кухарю може бути адресовано або ним створено множину повідомлень, але кожне таке повідомлення може бути створено або стосуватися тільки одного кухаря.

Таблиця `CookDelivery` представляє службу доставки, що застосовується цим кухарем для відправлення власних страв, використовуючи такі поля таблиці:

- `id` – номер;
- `cookid` – кухар, який користується цією службою доставки, представлений своїм номером;
- `deliveryid` – служба доставки, яка використовується для відправлення страв цим кухарем, представлена її номером;
- `price` – опис правил визначення вартості доставки.

Поле `cookid` таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею `Cook`, визначаючи відношення один до багатьох: кухар може використовувати множину служб доставки, представлену відповідними записами, але кожен такий запис стосується тільки одного кухаря.

Поле `deliveryid` таблиці – це зовнішній ключ, що сполучає цю таблицю з таблицею `Delivery`, визначаючи відношення один до багатьох: служба доставки може використовуватися множиною кухарів, представленою відповідними записами, але кожен такий запис стосується тільки однієї служби доставки.

Таблиця `Delivery` представляє службу або засіб доставки, використовуючи такі поля таблиці:

- `id` – номер;
- `named` – назва служби доставки або засіб доставки.

Результуюча структура БД приведена на рис. 2.3.

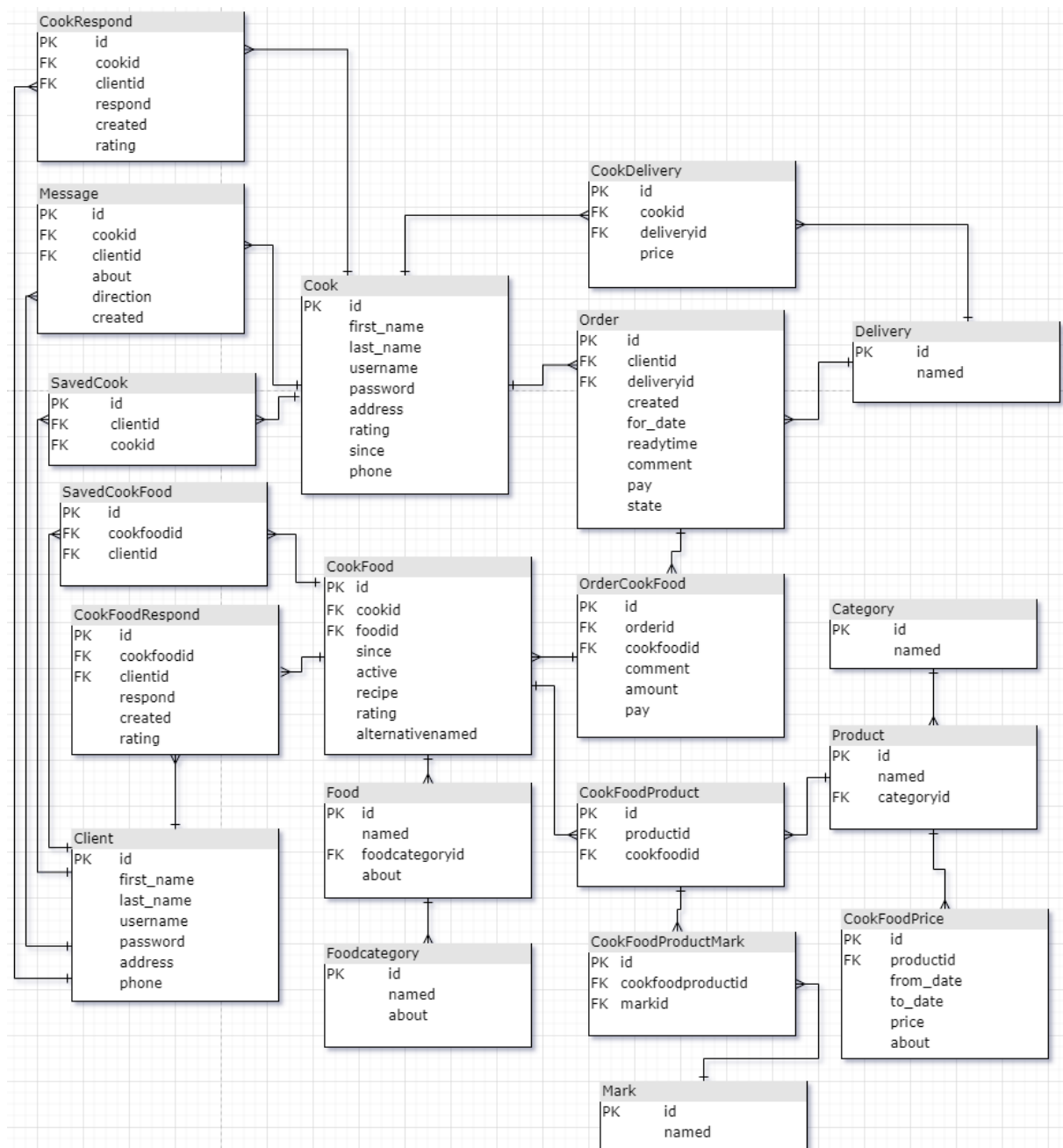


Рисунок 2.3 – Схема БД

2.4 Висновки за розділом 2

Обґрунтовано вибір мови Python для реалізації програми, що було розширено застосуванням вебфреймворку Django та системи керування БД PostgreSQL. Моделювання сценаріїв роботи клієнтів і кухарів з програмою мало діаграми прецедентів як результат цієї роботи. Детально описано створені таблиці і зв'язки між ними в структурі БД.

3 РОЗРОБКА ПРОГРАМИ

3.1 Робота з замовленнями їжі домашнього приготування

Порядок роботи з замовленнями визначає логіку організації різних частин програми. Для замовлення визначено ряд станів (рис. 3.1).

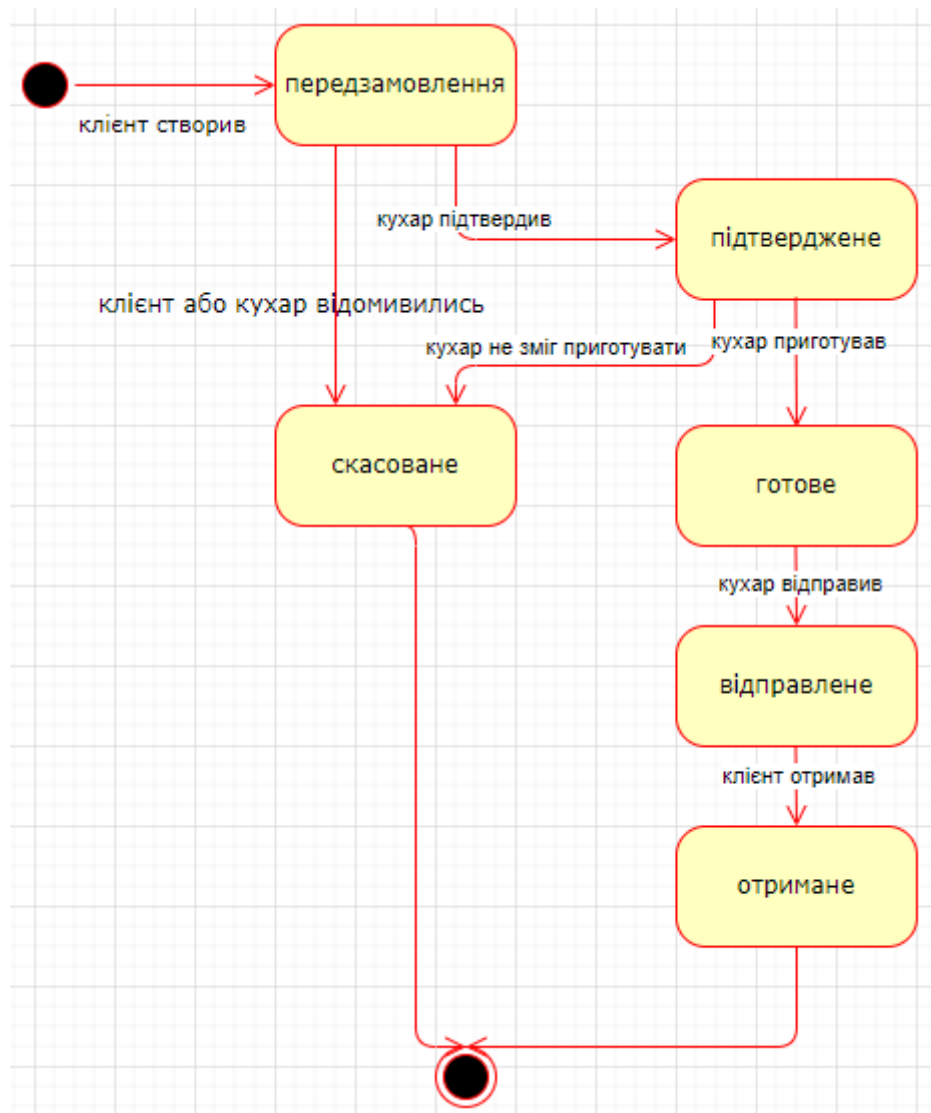


Рисунок 3.1 – Діаграма станів замовлення їжі домашнього приготування

Спершу після створення замовлення перебуває в стані передзамовлення. Після цього клієнт або кухар може його перевести до стану скасованого або кухар – до стану підтвердженого. Підтверджене замовлення кухар може перевести до стану готового, коли страви вже приготовлені. Далі

замовлення переходить до стану відправленого, а коли його отримав клієнт, то він переводить його до стану отриманого.

3.2 Опис реалізації програми

У відповідності з вебфреймворком Django програма складається з моделей, представлень, шаблонів сторінок і інших додаткових складових.

Представлення `OrderView` реалізує перегляд деталей замовлення, коли відкривається окрема сторінка, створена заповненням шаблону `homemadeorder.html` даними замовлення.

Представлення `OrderAddView` реалізує створення замовлення через підтримку роботи з відповідною формою. При цьому в методі `get_context_data` заповнюються дані, необхідні для заповнення форми, адже для підтримки роботи форми необхідні дані всіх служб доставки, які використовуються кухарем, з дому якого замовляються страви, а також список страв, з яких може обиратися замовлення. Оскільки в замовленні можуть бути страви тільки одного кухаря, то заходячи зі сторінки кухаря, фіксується в параметрі запиту `cook` номер кухаря, а далі за цим кухарем заповнюються всі доступні для нього служби і способи доставки, а також страви, які він готує:

```
kwargs['deliveries'] = CookDelivery.objects.filter(cookid =  
cook)  
kwargs['cookfoods'] = CookFood.objects.filter(cookid = cook)
```

У той же час для замовлення при його створенні важливим є метод підтримки роботи форми `form_valid`, адже у ньому зберігаються до об'єкту замовлення дані про службу доставки, яку обрав клієнт, дані про клієнта, виходячи з того облікового запису, що він використав для авторизації, дата створення замовлення, а також окремо обробляються страви, які були внесені в це замовлення. Страви необхідно обробляти окремо, тому що їх перелік

впливає на склад замовлення, з одного боку, а з іншого боку – на вартість замовлення загалом. Перелік страв з замовлення передається через параметр `foods` POST-запиту. Переданий перелік має складатися з номерів страв у виконанні кухаря з розділенням їх комами. Тоді далі цей перелік розділяється, всі страви перетворюються на відповідні об'єкти `CookFood`, для кожного такого об'єкта встановлюється його вартість на поточний момент, а далі знаходиться сума всіх цих страв у замовленні. Страва відповідно заноситься в замовлення створенням об'єкта `OrderCookFood`:

```
foods = str(self.request.POST.get('foods'))
foods = foods.split(',')
num = str(self.request.POST.get('nums'))
nums = num.split(',')
ofl = []
paysum = 0
i = 0
for f in foods:
    cf = CookFood.objects.get(id = f)
    price = CookFoodPrice.objects.filter(cookfoodid = cf,
from_date__lte = self.for_date, to_date__gte = self.for_date)
    am = nums [i]
    i = i + 1
    paysum = paysum + am * price.pay
    newordercf = OrderCookFood(cookfoodid = cf, orderid__id =
self.id, active = True, amount = am)
    newordercf.save()
form.instance.pay = paysum
```

Представлення `OrderClientBaseUpdateView` реалізує підтримку роботи з формою оновлення замовлення клієнтом на ранньому етапі, тобто поки кухар замовлення не підтвердив. При роботі представлення використовуються поля дати, на коли потрібна страва, коментаря до замовлення, служби доставки, що аналогічно попередньому уявленню створення замовлення, а також переліку страв у виконанні кухаря з замовлення. Цей перелік як і в попередньому уявленні передається через параметр `foods` запиту, а далі розбирається на частини, проте даний процес

відрізняється тим, що дані з переліку не просто вносяться в БД, вони спочатку аналізуються на наявність у БД в даний момент. Якщо вони вже є, то вони не змінюються. Якщо їх, тобто таких страв у складі замовлення немає, то вони додаються до складу замовлення в БД. Якщо ж вони були у замовленні, а тепер зникли, то відповідний запис з БД вилучається.

Також необхідно вказати, що оброблення форми відбувається тільки тоді, коли замовлення знаходиться в стані передзамовлення. Якщо замовлення знаходиться в більш пізньому стані, то оновлення його не дозволяється.

Представлення `OrderClientGetUpdateView` реалізує оновлення клієнтом власного замовлення в момент, коли він отримав його. При цьому змінюється стан замовлення і перевіряється, що замовлення було попередньо надіслано вже.

Представлення `OrderClientCancelView` реалізує скасування замовлення клієнтом у випадку, коли замовлення знаходилося в стані передзамовлення. Тоді стан змінюється, а також може змінюватися коментар до замовлення.

Представлення `OrderCookUpdateToAcceptView` реалізує оновлення замовлення кухарем, зі страв якого створено це замовлення. Оновлюються поля дати, на яку має бути готове замовлення, коментаря до замовлення, служби доставки та часу, коли замовлення готове. Кухар також може змінити перелік страв у замовленні. Це необхідно на випадок, якщо виявилось, що приготувати страву кухар не зможе. Проте в такому випадку страва з замовлення не вилучається, адже тоді клієнт може це не зрозуміти, а знімається її активність в замовленні:

```
for o in ocf:
    if o.cookfoodid not in ofl:
        ocfobj = self.ordercookfoods.filter(id = o.id)
        ocfobj.active = False
        ocfobj.save()
```

Автоматично також змінюється стан замовлення на наступний.

Представлення `OrderCookUpdateToReadyView` реалізує підтримку оновлення замовлення кухарем, коли страви вже приготовлені. Оновлюються поля служби доставки та часу, коли замовлення готове. Кухар також може змінити перелік страв у замовленні. Це необхідно на випадок, якщо виявилось, що приготувати страву кухар не зміг. Тоді як і в попередньому випадку деактивується відповідна страва з замовлення. Також автоматично оновлюється стан замовлення.

Представлення `OrderCookUpdateToReadyView` реалізує оновлення замовлення, яке вже приготовлене, в такому випадку його стан змінюється на надісланий автоматично.

Таким чином, кухар не може довільно переводити замовлення між станами, представлення підтримують окремо кожен етап процесу роботи над замовленням.

Представлення `ClientOrdersView` реалізує формування переліку замовлень клієнта, коли відкривається окрема сторінка, створена заповненням шаблону `clientorders.html` даними замовлень авторизованого у даний момент клієнта. Усі замовлення впорядковуються за датою створення, виводячи спершу ті, що були створені останніми.

Представлення `CookOrdersView` реалізує формування переліку замовлень, які стосуються страв авторизованого на даний момент кухаря. При цьому відкривається окрема сторінка, створена заповненням шаблону `cookorders.html`. Усі замовлення впорядковуються за датою створення, виводячи спершу ті, що були створені останніми. Окрім того в якості параметра запиту передається значення `state`, що визначає стан замовлень, які необхідно відобразити. Якщо значення не задано, то повертаються всі замовлення кухаря.

Представлення `CookFoodView` реалізує перегляд деталей страви кухаря, коли відкривається окрема сторінка, створена заповненням шаблону `homemadecookfood.html` даними цієї страви. Додатково при цьому надається інформація про поточну вартість страви.

Представлення `CookFoodAddView` реалізує створення нової страви у кухаря через підтримку роботи з відповідною формою. При цьому в методі `get_context_data` заповнюються дані, необхідні для заповнення форми, адже для підтримки роботи форми необхідні дані всіх страв, наявних у довіднику. Кухар може створити і власну страву, але стандартно за основу пропонується використовувати страву з довідника.

Однак страва кухаря складається не тільки з загальної інформації, але і з продуктів, які він використовує для її приготування. Продукти передаються окремо через параметр запиту `products`. Далі всі ці продукти аналізуються, розділяються через коми, виділяючи номери продуктів, і зберігаються в БД записи про відповідність кожного продукту відповідній страві кухаря:

```
products = str(self.request.POST.get('products'))
products = products.split(',')
for prod in products:
    p = Product.objects.get(id = prod)
    cookfp = CookFoodProduct(productid = p, cookfoodid__id =
self.id)
    cookfp.save()
```

Представлення `CookFoodUpdateView` реалізує підтримку роботи з формою оновлення страви кухаря. При цьому в методі `get_context_data` заповнюються дані, необхідні для заповнення форми, адже для підтримки роботи форми необхідні дані всіх страв, наявних у довіднику. Також при підтримці роботи форми в методі `form_valid` перевіряється перелік продуктів. Якщо якийсь продукт не було внесено до складу страви, то він додається, якщо ж він був у складі страви, але в оновленні зник, то відповідний запис з БД вилучається повністю.

Представлення `CookFoodForCookView` реалізує формування переліку страв, які готує кухар, створена заповненням шаблону `cookfoodforcook.html`

даними страв кухаря, які є активними на даний момент. Усі страви впорядковуються за рейтингом.

Представлення `CookFoodRespondAddView` реалізує створення відгуку про страву кухаря через підтримку роботи з відповідною формою. При цьому в методі `form_valid` при обробленні форми встановлюється клієнт, який створив цей відгук на основі даних авторизованого у даний момент користувача. Також визначається автоматично дата як поточна. Також визначається автоматично середня оцінка цієї страви кухаря на основі всіх відгуків за цією стравою:

```
cfrs = CookFoodRespond.objects.filter(cookfoodid =
self.cookfoodid)
total = 0
numb = cfrs.count()
for cfr in cfrs:
    total = total + cfr.rating / numb
cf = self.cookfoodid
cf.rating = total
```

Представлення `CookFoodRespondsView` реалізує формування переліку відгуків про страви, які готує кухар, створена заповненням шаблону `cookfoodresponds.html` даними відгуків про страву кухаря, яку передано в якості параметра в запиті.

Представлення `CookFoodSaveView` реалізує збереження страви кухаря для швидкого доступу клієнтом. При цьому в методі `form_valid` дані клієнта заповнюються автоматично на основі даних авторизованого користувача.

Представлення `CookFoodUnsaveView` реалізує вилучення страви кухаря з переліку швидкого доступу клієнта.

Представлення `CookFoodProductMarkDeleteView` реалізує вилучення позначки у продукта, з якого готується страва кухарем.

Представлення `SavedCookFoodsView` реалізує формування переліку страв кухаря, які були збережені для швидкого доступу, створена заповненням шаблону `savedcookfoods.html` даними збережених страв для

клієнта, який авторизований в системі. Нові збереження з'являються в потоці першими.

Представлення CookFoodSearchView реалізує підтримку пошуку страв у виконанні кухарів, що задовольняють критеріям пошуку. До критеріїв пошуку віднесено:

- мінімальний рейтинг страви кухаря – параметр пошуку `minrating`;
- максимальний рейтинг страви кухаря – параметр пошуку `maxrating`;
- конкретна страва – параметр пошуку `food`;
- категорія страви – параметр пошуку `category`;
- перелік позначок продуктів, яким усі продукти в страві повинні задовольняти, – параметр пошуку `marks`;
- перелік продуктів, які повинні входити до страви, – параметр пошуку `products`.

Значення параметрів можуть бути пропущені.

Окрім того заповнюється параметр `allmarks`, який містить для роботи форми перелік всіх можливих позначок для продуктів.

Також заповнюється параметр `allproducts`, який містить перелік усіх продуктів. Ще заповнюється параметр `allfcategories`, який містить перелік усіх категорій страв, параметр `allcategories`, який містить перелік всіх категорій продуктів, а також параметр `allfoods`, який містить перелік всіх страв.

Сам пошук після виділення списку продуктів та позначень з параметрів пошуку виконується наступним чином.

Спочатку аналізується, чи входять до переліку продуктів страв заданий продукт, і якщо не входить, то зберігається примітка про необхідність вилучення цієї страви з результатів пошуку:

```
prodsres = True
for p in prods:
    if not (p in prods):
        prodsres = False
```

Також перевіряється, що кожен продукт у страві має відповідну встановлену під час пошуку позначку. Якщо хоча б один продукт такої позначки не містить, то страва вилучається з переліку:

```
for p in prods:
    res = True
    marks = cookfoodproduct_marks
    for mark in allmarks:
        if not (mark in marks):
            res = False
            prodsres = False
```

Представлення CookFoodPriceAddView реалізує створення запису про вартість страви у кухаря в певний період часу.

Представлення CookFoodPriceUpdateView реалізує оновлення запису про вартість страви у кухаря в певний період часу.

Представлення CookFoodProductMarkAddView реалізує створення запису про позначку для продукту в складу страви, яку готує кухар.

Клас Order реалізує модель замовлення за допомогою полів:

- clientid – поле клієнта, який створив замовлення, є полем зовнішнього ключа від моделі Client;
- created – поле дати створення замовлення, є полем дати і часу;
- for_date – поле дати і часу, на який повинно бути готове замовлення, є полем дати і часу;
- comment – поле коментаря до замовлення, є символьним полем;
- pay – поле суми, яку необхідно сплатити за замовлення, є числом з фіксованою точністю;
- deliveryid – поле доставка, яка буде використовуватися для доставки замовлення, представлена її номером, є полем зовнішнього ключа від моделі Delivery;
- state – поле стану замовлення, що є символьним полем, значення якого обмежено створеним набором станів.

Клас OrderCookFood реалізує модель замовлення за допомогою полів:

- orderid – поле замовлення, до складу якого входить ця страва, є полем зовнішнього ключа від моделі Order;
- cookfoodid – поле страви авторства певного кухаря, що входить до замовлення, є полем зовнішнього ключа від моделі CookFood;
- comment – поле коментаря до особливостей страви у цьому замовленні, є полем символьного типу;
- amount – поле кількості одиниць цієї страви у замовленні, є полем цілочисельного типу;
- pay – поле вартості одиниці страви на момент замовлення, є числом з фіксованою точністю.

Метод save моделі реалізує обчислення вартості цієї страви на момент, коли має бути готове замовлення, вибираючи відповідне значення з БД:

```
price = CookFoodPrice.objects.filter(cookfoodid =
self.cookfoodid, from_date__lte = self.orderid.for_date, to_date__gte
= self.orderid.for_date)
self.pay = price.pay
```

Клас Cook реалізує модель сутності кухаря наслідуванням класу AbstractUser, представляючи кухаря за допомогою полів:

- address – поле адреси, представлене текстовим полем;
- rating – поле рейтингу кухаря, представлене полем числа з обмеженою точністю;
- since – поле дати, з якої кухар почав працювати з системою, представлене полем дати і часу;
- phone – поле номеру телефону кухаря, представлене полем символьного типу.

Клас Client реалізує модель сутності клієнта наслідуванням класу AbstractUser, представляючи клієнта за допомогою полів:

- address – поле адреси клієнта, яка стандартно використовується для доставки, є текстовим полем;

- phone – поле номеру телефону клієнта, є символьним полем.

Клас Food реалізує модель страви за допомогою полів:

- named – поле назви страви, є полем символьного типу;

- foodcategoryid – поле категорії страви, є полем зовнішнього ключа від моделі FoodCategory;

- about – поле опису страви, є текстовим полем.

Клас CookFood реалізує модель страви кухаря за допомогою полів:

- cookid – поле кухаря, який готує цю страву, є полем зовнішнього ключа від моделі Cook;

- foodid – поле страви, яку готує кухар, є полем зовнішнього ключа від Food;

- alternativenamed – поле альтернативної назви страви у цього кухаря або назва страви, якої немає в довіднику страв, є полем символьного типу;

- since – поле дати, з якої цей кухар готує цю страву, надаючи про це дані в системі, є полем дати і часу;

- active – поле, яке визначає, чи це активний запис, чи ні, є полем логічного типу;

- recipe – поле особливостей приготування страви у цього кухаря, є полем текстового типу;

- rating – поле рейтингу страви у виконанні цього кухаря, є полем числа з обмеженою точністю.

Клас Foodcategory реалізує модель категорії страви за допомогою полів:

- named – поле назви категорії страв, є полем символьного типу;

- about – поле опису категорії страв, є полем текстового типу.

Клас Category реалізує модель категорії страви за допомогою поля named, що є полем назви категорії продукту, яке є символьним полем.

Клас Product реалізує модель продукту за допомогою полів:

- `named` – поле назви продукту, є символьним полем;
- `categoryid` – поле категорії продукту, є полем зовнішнього ключа моделі `Category`.

Клас `Mark` реалізує модель позначки за допомогою поля `named`, що є назвою позначки про продукт, є символьним полем.

Клас `CookFoodProduct` реалізує модель продукту, з якого виготовляється страва, за допомогою полів:

- `productid` – поле продукту, що входить до складу страви, є полем зовнішнього ключа від моделі `Product`;
- `cookfoodid` – поле реалізації страви кухарем, в якій використовується цей продукт, є полем зовнішнього ключа від моделі `CookFood`.

Клас `CookFoodProductMark` реалізує модель позначки, встановленої продукту, з якого виготовляється страва, за допомогою полів:

- `cookfoodproductid` – поле продукту, з якого виготовляється страва, є полем зовнішнього ключа від моделі `CookFoodProduct`;
- `markid` – поле позначки, встановленої цьому продукту, є полем зовнішнього ключа від моделі `Mark`.

Клас `CookFoodPrice` реалізує модель вартості страви у виконанні кухаря, встановленої продукту, з якого виготовляється страва, за допомогою полів:

- `cookfoodid` – номер страви у виконанні певного кухаря, якій встановлюється вартість, є полем зовнішнього ключа від моделі `CookFood`;
- `from_date` – поле дати, з якої встановлена ця вартість страви, є полем дати і часу;
- `to_date` – поле дати, до якої встановлена ця вартість страви, є полем дати і часу;
- `pay` – поле вартості страви у цей період, є полем дати і часу;
- `about` – поле особливостей страви у цей період, є текстовим полем.

Клас `CookFoodRespond` реалізує модель відгук про страву у виконанні кухаря за допомогою полів:

- cookfoodid – поле страви у виконанні певного кухаря, якої стосується відгук, є полем зовнішнього ключа моделі CookFood;

- clientid – поле клієнта, який створив цей відгук, є полем зовнішнього ключа моделі Client;

- respond – поле відгуку, є текстовим полем;

- created – поле дати створення відгуку, поле дати і часу;

- rating – поле оцінки у відгуку, є полем числа з обмеженою точністю.

Клас CookRespond реалізує модель відгук про кухаря за допомогою полів:

- cookid – поле кухаря, про якого створено відгук, є зовнішнім полем з моделі Cook;

- clientid – поле клієнта, який створив відгук, є зовнішнім полем з моделі Client;

- respond – поле відгуку, є текстовим полем;

- created – поле дати, коли було створено відгук про кухаря, є полем дати і часу;

- rating – поле оцінки у відгуку, є полем числа з обмеженою точністю.

Клас SavedCook реалізує модель збережених для швидкого доступу даних кухаря за допомогою полів:

- clientid – поле клієнта, який зберіг дані кухаря, є зовнішнім ключем моделі Client;

- cookid – поле кухаря, якого зберіг клієнт, є зовнішнім ключем моделі Cook.

Клас SavedCookFood реалізує модель збереженої для швидкого доступу страви у виконанні кухаря за допомогою полів:

- clientid – поле клієнта, який зберіг цю страву, є зовнішнім ключем моделі Client;

- cookfoodid – поле страви у виконанні певного кухаря, яку зберіг клієнт, є зовнішнім ключем моделі CookFood.

Клас Message реалізує модель повідомлення між кухарем і клієнтом за допомогою полів:

- clientid – поле клієнта, який створив або якому адресовано повідомлення, є зовнішнім ключем моделі Client;
- cookid – поле кухаря, який створив або якому адресовано повідомлення, є зовнішнім ключем моделі Cook;
- direction – поле напрямку повідомлення, є полем логічного типу;
- about – поле тексту повідомлення, є полем текстового типу;
- created – поле дати створення повідомлення, є полем дати і часу.

Клас CookDelivery реалізує модель служби доставки, що застосовується цим кухарем для відправлення власних страв, за допомогою полів:

- cookid – поле кухаря, який користується цією службою доставки, є зовнішнім полем моделі Cook;
- deliveryid – поле служби доставки, що використовується для відправлення страв цим кухарем, є зовнішнім полем моделі Delivery;
- price – поле опису правил визначення вартості доставки, є текстовим полем.

Клас Delivery реалізує модель служби доставки за допомогою поля named, що є полем назви служби доставки, представленим символьним полем.

3.3 Висновки за розділом 3

За результатами розділу представлено опис станів, в яких може перебувати замовлення в програмі, на основі діаграми станів, виконано реалізацію програми та описано основні моменти стосовно виконаної реалізації.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма призначена для підтримки людей, які хочуть харчуватися їжею домашнього приготування, та людей, які готові готувати вдома таку їжу і заробляти за це кошти. Програма надає для цього засоби розширеного пошуку страв клієнтам, засоби внесення розширеної інформації про страви, які готує кухар вдома, та засоби підтримки роботи з замовленнями їжі домашнього приготування від клієнтів до кухарів.

4.2 Умови виконання програми

Технічні засоби в мінімальній конфігурації, достатній для роботи програми, охоплюють 1 Гб пам'яті жорсткого диску, 4 Гб оперативної пам'яті, 4 ядра процесору (частота 2,8 ГГц).

4.3 Тестування і опис принципів експлуатації програми

Тестування програми виконувалось на основі перевірки реалізації функцій шляхом співставлення очікуваних результатів роботи і відображуваних результатів. Це дозволило встановити в результаті повну відповідність і вважати результати тестування успішними.

Отримані під час зокрема тестування інтерфейси програми при виконанні цих функцій були використані для ілюстрації роботи з програмою в подальшому.

Одразу потрібно підкреслити, що програма має окремі інтерфейси для роботи клієнта і кухаря.

Кухар може зі своїх сторінок звертатися до різних доступних йому засобів. Усі вони об'єднані в розділи «Страви», «Замовлення», «Відгуки», «Переписка», «Профіль».

Для того щоб визначити перелік власних страв кухаря, потрібно звернутися до розділу «Страви» (рис. 4.1).

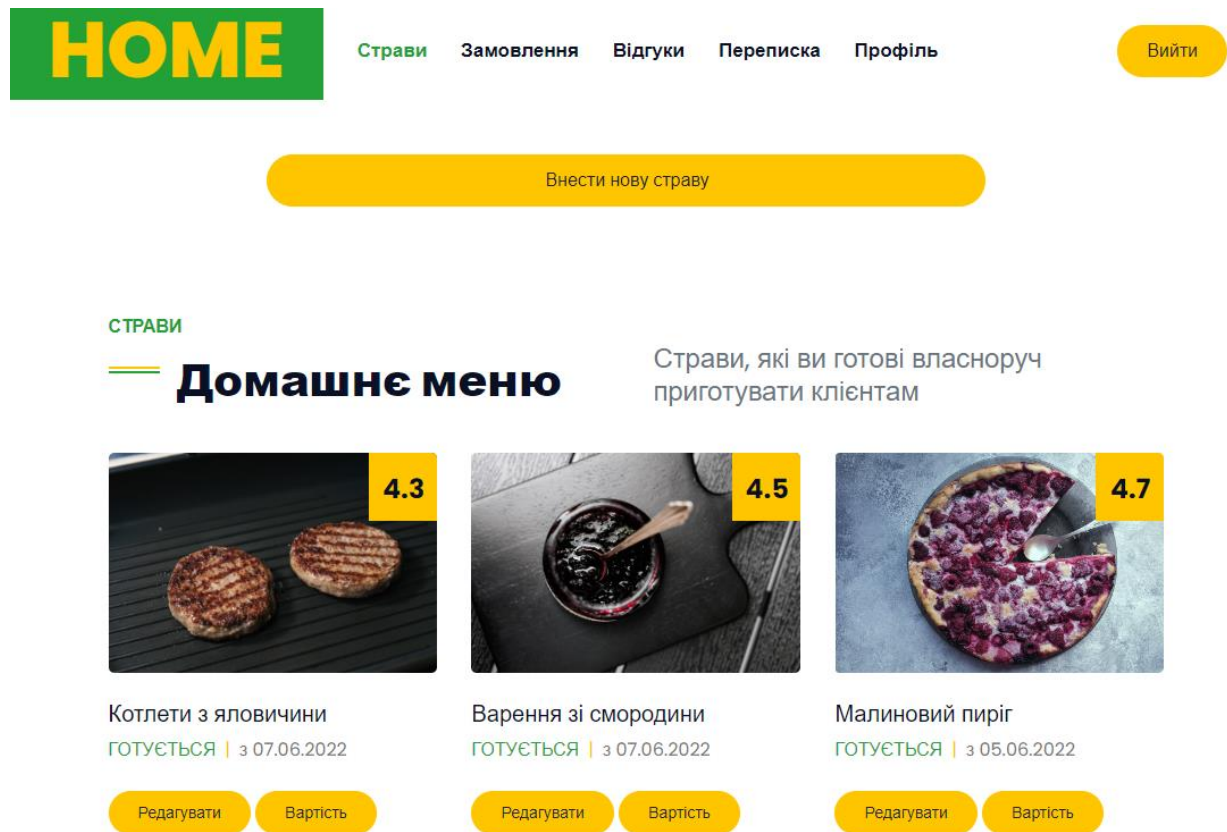


Рисунок 4.1 – Сторінка керування власними стравами

На сторінці керування власними стравами доступний перелік всіх страв кухаря, включаючи як активні, так і неактивні. Це абсолютно всі його страви. У свою чергу клієнту зі сторінки кухаря доступні тільки активні страви, тобто ті страви, які кухар продовжує готувати.

Для кожної страви показується її фотографія, середня оцінка, виставлена клієнтами, які її замовляли, назва страви, стан (активний, що відображається написом «готується», чи неактивний), дата з якої страва готується через систему.

Також кнопки керування дозволяють перейти до повноцінного редагування всіх даних страви, включаючи зокрема перелік продуктів, позначки, встановлені цим продуктам, а також всю загальну інформацію.

При цьому слід звернути увагу, що відображається назва страви, яку затвердив кухар. Тобто це може бути тарт, а кухар йому встановив назву «Полуничний тарт». Таким чином кухар може створювати і страви, назв яких немає в довіднику.

Кнопка «Вартість» надає можливість перейти до встановлення вартості страви на різні періоди часу.

З розділу «Замовлення» кухарю доступні можливості керування всіма замовленнями, що стосуються його страв (рис. 4.2).

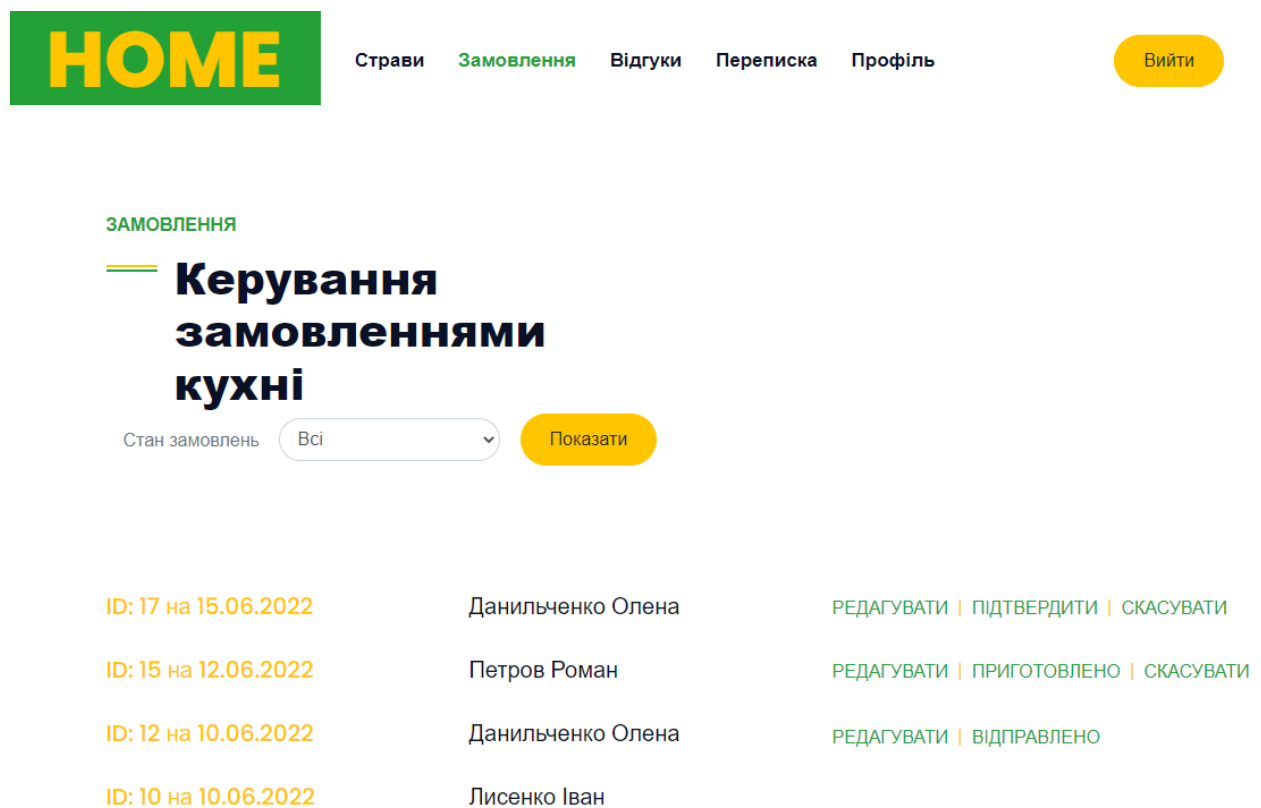


Рисунок 4.2 – Сторінка керування замовленнями

Для кожного замовлення відображається його номер та дата, на яку воно має бути готове. Якщо натиснути на це посилання, то відкриється сторінка з повною інформацією про замовлення. Також за замовленням відображається прізвище та ім'я клієнта, а також кнопки для керування замовленнями. Кнопки змінюються в залежності від стану замовлення. Спочатку доступні кнопки для редагування повного замовлення, для

підтвердження замовлення або його скасування. Після цього замовлення можна відредагувати, змінивши перелік страв, а після зберігши його, визначивши, що замовлення приготовлено, або за допомогою кнопки «Приготовлено» визначити готовність замовлення. Також замовлення в цей момент можна скасувати. У подальшому замовлення можна або відредагувати, або визначити, що воно відправлено. Після цього у кухаря більше немає засобів керування замовленнями, стан такого замовлення може змінювати тільки клієнт.

З розділу «Переписка» кухарю доступна взаємодія зі всіма клієнтами, розділеними за окремими посиланнями, а за кожним посиланням відкривається відповідний перелік повідомлень з можливістю відправити власне.

З розділу «Відгуки» кухар може отримати перелік всіх відгуків, які про нього створили клієнти. Це відгуки саме про кухаря. Відгуки клієнта про страви кухаря доступні зі сторінки відповідної страви.

Клієнт має свої засоби керування роботою з програмою. Йому доступні розділи «Пошук», «Замовлення», «Переписка», «Швидкодія», «Профіль».

З розділу «Пошук» клієнт може виконати пошук потрібних йому страв. Ця сторінка відкривається одразу після звернення до вебсайту в режимі клієнта.

Під час пошуку страв (рис. 4.3) клієнт може задати наступні параметри:

- мінімальний рейтинг страви;
- максимальний рейтинг страви;
- конкретну страву, для чого потрібну спочатку задати категорію страви, а потім саму страву;
- категорію страви – для ситуацій, коли немає необхідності визначати конкретну страву;

- перелік позначок продуктів, яким усі продукти в страві повинні задовольняти: для цього потрібно обрати з переліку позначку, а тоді натиснути на кнопку «Додати», тоді нижче червоним кольором з’явиться відповідна позначка;
- перелік продуктів, які повинні входити до страви: працює аналогічно до позначок.

ДОМАШНІ СМАКОЛИКИ

Пошук страв

Мінімальний рейтинг

3 ▼

Максимальний рейтинг

5 ▼

Категорія страви

Будь-яка ▼

Страва

Будь-яка ▼

Характеристики продуктів

Сезонні ▼

Додати

Додано: Екологічні

Продукти

Апельсин ▼

Додати

Додано: Полуниця

Шукати

Рисунок 4.3 – Сторінка пошуку страв

Після визначення параметрів пошуку натискається кнопка «Шукати», і відображаються результати пошуку (рис. 4.4). Кожен результат включає фотографію, назву страви, нік кухаря, середню оцінку, перше речення опису,

а також кнопки. Кнопка «Відкрити» дозволяє відкрити сторінку зі стравою, звідки можна також перейти до її замовлення. Також відкрити сторінку можна, натиснувши на фотографію страви. Кухар може відкрити страву, натискаючи тільки на фотографію, для нього окремої кнопки не виділено. Також за кожною стравою виділяється її вартість на поточний момент, а також надається кнопка «Замовлення» для початку формування замовлення, коли відкривається сторінка, на якій вже додано одну страву, а потім можна так само, як це було показано в пошуці, додавати до замовлення інші страви.

СТРАВИ

Результати пошуку

 300 ГРН	 230 ГРН	 280 ГРН
Полуничний пиріг HONOR Оцінка: 4.8	Чизкейк MANMAN Оцінка: 4.8	Полуничний тарт HONOR Оцінка: 4.7
Полуничний пиріг на тонкому коржіку з додаванням голубики	Класичний чизкейк з додаванням ягід полуниці	Класичний тарт вкритий товстим шаром полуниці
Відкрити Замовлення	Відкрити Замовлення	Відкрити Замовлення

Рисунок 4.4 – Результати пошуку страв домашнього приготування

Натискаючи на нік кухаря, можна перейти до сторінки кухаря з відгуками про нього.

З розділу «Замовлення» клієнт може керувати тільки власними замовленнями.

З розділу «Переписка» клієнту доступна взаємодія зі всіма кухарями, розділеними за окремими посиланнями, а за кожним посиланням

відкривається відповідний перелік повідомлень з можливістю відправити власне.

З розділу «Швидкодія» можна отримати перелік кухарів та окремо перелік страв, які зберіг клієнт для швидкого доступу. Звідси можна також вилучити ці записи за допомогою відповідного посилання. Додати ж страви або кухарів можна зі сторінки відповідної страви або кухаря, там доступна кнопка «Швидкий доступ».

4.4 Висновки за розділом 4

Виконане тестування роботи кухаря і клієнта з програмою відбулось успішно. В розділі описано процес роботи цих користувачів з програмою.

ВИСНОВКИ

У дипломній роботі успішно досягнуто мету розроблення ПЗ замовлення їжі домашнього приготування для встановлення взаємодії між кухарями і клієнтами.

Обґрунтовано вибір мови Python для реалізації програми, що було розширено застосуванням вебфреймворку Django та системи керування БД PostgreSQL. Моделювання сценаріїв роботи клієнтів і кухарів з програмою мало діаграми прецедентів як результат цієї роботи. Детально описано створені таблиці і зв'язки між ними в структурі БД.

Базовою сутністю виділено замовлення, для якого визначено стани, що лягли в основу проєктування програми, включаючи структури даних, структуру сторінок, представлень програми, порядок взаємодії з програмою. Стани представлено діаграмою станів. Описано детально, як було реалізовано основні структурні одиниці програми, включаючи моделі програми, її представлення.

Програма реалізована як вебзастосунок, що надає можливості визначення страв, які готує кухар, створення відгуків про страви у кухаря і про кухаря, замовлення домашніх страв у кухаря, керування замовленнями страв, пошук домашніх страв, перегляд замовлень клієнтом, перегляд відгуків про кухаря, збереження страв кухаря у власний перелік для швидкого доступу, а даних кухаря у власний перелік для швидкого доступу, надсилання повідомлень, відтворення переписки, перегляду страв кухаря.

Розроблена програма дозволяє підтримувати роботу бажаючих готувати вдома для інших людей шляхом організації замовлень таких страв, керування цими замовленнями, підтримувати клієнтів, які бажають визначити те, які страви прийдуть їм до смаку і знайти серед цього різноманіття саме те, що дозволить харчуватися домашньою і до того ж можливо здоровою їжею.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Cooking at home becomes major trend coming out of Covid-19 [Electronic resource]. – Access mode : <https://www.consultancy.uk/news/25412/cooking-at-home-becomes-major-trend-coming-out-of-covid-19>.
2. Devenyns, J. Survey [Electronic resource] : 7 in 10 consumers say they will keep cooking at home after the pandemic / J. Devenyns. – Access mode : <https://www.fooddive.com/news/survey-7-in-10-consumers-say-they-will-keep-cooking-at-home-after-the-pand/593532/>.
3. 6 Benefits of Homemade Meals + 7 Recipes [Electronic resource]. – Access mode : <https://thewholeu.uw.edu/2015/03/06/homemade-meals/>.
4. HomeMade [Electronic resource] : Find local, fresh and healthy homemade food delivered to you. – Access mode : <https://homemadefood.app/>.
5. HomeMade for Home Cooks on the App Store [Electronic resource]. – Access mode : <https://apps.apple.com/us/app/homemade-for-home-cooks/id1470576517>.
6. Maids, V. 7 Apps To Get Home Cooked Food To Your Doorstep [Electronic resource] / V. Maids. – Access mode : <https://valetmaids.com/7-apps-to-get-home-cooked-food-to-your-doorstep/>.
7. Novikova, A. 5 Best Homemade Food Delivery Apps for the USA (Android & iOS) [Electronic resource] / A. Novikova. – Access mode : <https://freeappsforme.com/homemade-food-delivery-apps-usa/>.
8. Homemade Food Near You – Savorly [Electronic resource]. – Access mode : <https://savorly.app/>.
9. Savorly [Electronic resource] : Home Cooked Meals on Google Play. – Access mode : <https://play.google.com/store/apps/details?id=com.savorly&hl=uk&gl=US>.
10. Matthes, E. Python Crash Course [Text] : A Hands-On, Project-Based Introduction to Programming / E. Matthes. – San Francisco : No Starch Press,

2019. – 544 p.

11. Vincent, W. S. Django for Beginners [Text] : Build Websites with Python and Django (Welcome to Django) / W. S. Vincent. – Bristol : WelcomeToCode, 2020. – 317 p.

ДОДАТОК А
Технічне завдання

Вступ

Ресторанний бізнес у розвинутих країнах дуже активно просякнув життя громадян. Але домашнє харчування має свої особливості, дозволяє забезпечувати здорове харчування, дозволяє контролювати харчування, проте потребує і додаткових зусиль і часу.

Зараз для багатьох людей приготування їжі самостійно є проблемою, проте вони можуть бажати все одно харчуватися домашньою їжею. З іншого боку інші люди готові витратити час на приготування страв і заробляти за це гроші. Саме для створення контакту між ними і має використовуватися розроблювана програма.

А.1 Підстави для розробки

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

А.2 Призначення розробки

Програма призначена для підтримки людей, які хочуть харчуватися їжею домашнього приготування, та людей, які готові готувати вдома таку їжу і заробляти за це кошти. Програма має надавати для цього засоби розширеного пошуку страв клієнтам, засоби внесення розширеної інформації про страви, які готує кухар вдома, та засоби підтримки роботи з замовленнями їжі домашнього приготування від клієнтів до кухарів.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Функції, які виконуватиме програма за результатами розробки:

- визначення страв, які готує кухар;
- створення відгуків про страви у кухаря;
- створення відгуків про кухаря;
- замовлення домашніх страв у кухаря;
- керування замовленнями страв;
- пошук домашніх страв;
- перегляд замовлень клієнтом;
- перегляд відгуків про кухаря;
- збереження страв кухаря у власний перелік для швидкого доступу;
- збереження даних кухаря у власний перелік для швидкого доступу;
- надсилання повідомлень;
- відтворення переписки;
- перегляд страв кухаря;
- авторизація користувачів;
- реєстрація клієнтів і кухарів.

A.3.2 Умови експлуатації

Рекомендовано для забезпечення роботи програми виділити системного адміністратора, до зобов'язань якого віднести підтримку роботи вебсервера, сервера БД та внесення за потреби базових даних.

A.3.3 Вимоги до складу та параметрів технічних засобів

Технічні засоби в мінімальній конфігурації, достатній для роботи програми, охоплюють 1 Гб пам'яті жорсткого диску, 4 Гб оперативної

пам'яті, 4 ядра процесору (частота 2,8 ГГц).

А.4 Порядок контролю та приймання

Остаточна перевірка отриманих результатів виконання роботи здійснюється під час захисту дипломної роботи.

ДОДАТОК Б
Текст програми

Б.1 Текст файла `orderview.py`

```

from django.contrib.auth.mixins import LoginRequiredMixin
from datetime import datetime
from django.views.generic import ListView
from django.views.generic import UpdateView
from django.views.generic.edit import CreateView
from django.views.generic.detail import DetailView
from django.urls import reverse_lazy
from .models import Order, Delivery, CookFood, OrderCookFood,
CookDelivery, CookFoodPrice, Cook, Client

class OrderView (DetailView):
    template_name = 'homemadeorder.html'
    model = Order

class OrderAddView(LoginRequiredMixin, CreateView):
    model = Order
    fields = ['for_date', 'comment', 'deliveryid']
    template_name = 'orderadd.html'

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

    def get_context_data(self, **kwargs):
        self.deliveryid = get_object_or_404(Delivery,
id=self.kwargs['category_id'])
        kwargs['deliveryid'] = self.deliveryid
        cookid = self.kwargs['cook']
        cook = get_object_or_404(Cook, id = cookid)
        kwargs['deliveries'] =
CookDelivery.objects.filter(cookid = cook)
        kwargs['cookfoods'] = CookFood.objects.filter(cookid =
cook)

        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.deliveryid = get_object_or_404(Delivery,
id=self.kwargs['category_id'])
        client = self.request.user
        form.instance.clientid = client
        form.instance.pay = 0

```

```

        form.instance.created = datetime.today()
        foods = str(self.request.POST.get('foods'))
        foods = foods.split(',')
        num = str(self.request.POST.get('nums'))
        nums = num.split(',')
        i = 0
        ofl = []
        paysum = 0
        for f in foods:
            cf = CookFood.objects.get(id = f)
            price = CookFoodPrice.objects.filter(cookfoodid =
cf, from_date__lte = self.for_date, to_date__gte = self.for_date)
            am = nums [i]
            i = i + 1
            paysum = paysum + am * price.pay
            newordercf = OrderCookFood(cookfoodid = cf,
orderid__id = self.id, active = True, amount = am)
            newordercf.save()
        form.instance.pay = paysum
        form.instance.state = 'A'
        return super().form_valid(form)

```

```

class OrderClientBaseUpdateView(LoginRequiredMixin, UpdateView):
    model = Order
    fields = ['for_date', 'comment', 'deliveryid']
    template_name = 'orderclientupdate.html'

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

    def get_context_data(self, **kwargs):
        self.deliveryid = get_object_or_404(Delivery,
id=self.kwargs['category_id'])
        kwargs['deliveryid'] = self.deliveryid
        ocf = self.ordercookfoods
        foods = foods.split(',')
        food = foods[0]
        cook = CookFood.objects.get(id = food)
        cook = cook.cookid
        kwargs['deliveries'] =
CookDelivery.objects.filter(cookid = cook)
        return super().get_context_data(**kwargs)

    def form_valid(self, form):

```

```

        self.deliveryid = get_object_or_404(Delivery,
id=self.kwargs['category_id'])
        form.instance.deliveryid = self.deliveryid
        if self.state = 'A'
            foods = str(self.request.POST.get('foods'))
            ocf = self.ordercookfoods
            foods = foods.split(',')
            ofl = []
            num = str(self.request.POST.get('nums'))
            nums = num.split(',')
            i = 0
            paysum = 0
            for f in foods:
                cf = CookFood.objects.get(id = f)
                price =
CookFoodPrice.objects.filter(cookfoodid = cf, from_date__lte =
self.for_date, to_date__gte = self.for_date)
                am = nums[i]
                i = i + 1
                paysum = paysum + price.amount * am
                ofl.append(cf)
                of = OrderCookFood.objects.get(cookfoodid =
cf, orderid__id = self.id)
                if not of:
                    newordercf = OrderCookFood(cookfoodid
= cf, orderid__id = self.id)
                    newordercf.save()
            for o in ocf:
                if o.cookfoodid not in ofl:
                    self.ordercookfoods.filter(id =
o.id).delete()

            form.instance.pay = paysum
            return super().form_valid(form)

class OrderClientGetUpdateView(LoginRequiredMixin, UpdateView):
    model = Order
    fields = ['comment']
    template_name = 'orderclientgetupdate.html'

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

    def form_valid(self, form):
        if self.state = 'D'

```

```

        form.instance.state = 'E'
        return super().form_valid(form)

class OrderClientCancelView(LoginRequiredMixin, UpdateView):
    model = Order
    fields = ['comment']
    template_name = 'orderclientcancel.html'

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

    def form_valid(self, form):
        if self.state = 'A':
            self.state = 'F'
        return super().form_valid(form)

class OrderCookUpdateToAcceptView(LoginRequiredMixin,
UpdateView):
    model = Order
    fields = ['for_date', 'comment', 'deliveryid', 'readytime']
    template_name = 'ordercookaupdate.html'

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

    def get_context_data(self, **kwargs):
        self.deliveryid = get_object_or_404(Delivery,
id=self.kwargs['category_id'])
        kwargs['deliveryid'] = self.deliveryid
        cookid = self.request.user
        cook = get_object_or_404(Cook, id = cookid.id)
        kwargs['deliveries'] =
CookDelivery.objects.filter(cookid = cook)

        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.deliveryid = get_object_or_404(Delivery,
id=self.kwargs['category_id'])
        form.instance.deliveryid = self.deliveryid
        foods = str(self.request.POST.get('foods'))
        ocf = self.ordercookfoods
        foods = foods.split(',')

```

```

        num = str(self.request.POST.get('nums'))
        nums = num.split(',')
        i = 0
        ofl = []
        paysum = 0
        for f in foods:
            cf = CookFood.objects.get(id = f)
            price = CookFoodPrice.objects.filter(cookfoodid =
cf, from_date__lte = self.for_date, to_date__gte = self.for_date)
            am = nums [i]
            i = i + 1
            paysum = paysum + price.pay * am
            ofl.append(cf)
            of = OrderCookFood.objects.get(cookfoodid = cf,
orderid__id = self.id)
            if not of:
                newordercf = OrderCookFood(cookfoodid = cf,
orderid__id = self.id)
                newordercf.save()
        for o in ocf:
            if o.cookfoodid not in ofl:
                ocfobj = self.ordercookfoods.filter(id =
o.id)

                ocfobj.active = False
                ocfobj.save()
        self.state = 'B'
        form.instance.pay = paysum
        return super().form_valid(form)

class OrderCookUpdateToReadyView(LoginRequiredMixin,
UpdateView):
    model = Order
    fields = ['deliveryid', 'readytime']
    template_name = 'ordercookbupdate.html'

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

    def get_context_data(self, **kwargs):
        self.deliveryid = get_object_or_404(Delivery,
id=self.kwargs['category_id'])
        kwargs['deliveryid'] = self.deliveryid
        cookid = self.request.user
        cook = get_object_or_404(Cook, id = cookid.id)

```

```

        kwargs['deliveries'] =
CookDelivery.objects.filter(cookid = cook)
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.deliveryid = get_object_or_404(Delivery,
id=self.kwargs['deliveryid'])
        form.instance.deliveryid = self.deliveryid
        foods = str(self.request.POST.get('foods'))
        ocf = self.ordercookfoods
        foods = foods.split(',')
        num = str(self.request.POST.get('nums'))
        nums = num.split(',')
        i = 0
        ofl = []
        paysum = 0
        for f in foods:
            cf = CookFood.objects.get(id = f)
            price = CookFoodPrice.objects.filter(cookfoodid =
cf, from_date__lte = self.for_date, to_date__gte = self.for_date)
            am = nums[i]
            i = i + 1
            paysum = paysum + price.pay * am
            ofl.append(cf)
            of = OrderCookFood.objects.get(cookfoodid = cf,
orderid__id = self.id)
            if not of:
                newordercf = OrderCookFood(cookfoodid = cf,
orderid__id = self.id)
                newordercf.save()
        for o in ocf:
            if o.cookfoodid not in ofl:
                ocfobj = self.ordercookfoods.filter(id =
o.id)

                ocfobj.active = False
                ocfobj.save()
        self.state = 'C'
        form.instance.pay = paysum
        return super().form_valid(form)

class OrderCookUpdateToSentView(LoginRequiredMixin, UpdateView):
    model = Order
    fields = ['deliveryid', 'readytime']
    template_name = 'ordercookdupdate.html'

```

```

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

    def get_context_data(self, **kwargs):
        self.deliveryid = get_object_or_404(Delivery,
id=self.kwargs['category_id'])
        kwargs['deliveryid'] = self.deliveryid
        cookid = self.request.user
        cook = get_object_or_404(Cook, id = cookid.id)
        kwargs['deliveries'] =
CookDelivery.objects.filter(cookid = cook)
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.deliveryid = get_object_or_404(Delivery,
id=self.kwargs['deliveryid'])
        form.instance.deliveryid = self.deliveryid
        self.state = 'D'
        return super().form_valid(form)

class ClientOrdersView(ListView):
    model = Order
    template_name = 'clientorders.html'
    paginate_by = 10

    def get_queryset(self):
        client = self.request.user
        cl = Client.objects.get(id = client.id)
        orders = Order.objects.filter(clientid = cl)
        orders = orders.order_by('-created')
        return orders

class CookOrdersView(ListView):
    model = Order
    template_name = 'cookorders.html'
    paginate_by = 10

    def get_queryset(self):
        ck = self.request.user
        neccstate = self.request.GET.get('state')
        cook = Cook.objects.get(id = ck.id)
        if neccstate:
            orders = Order.objects.filter(state = neccstate)
        else:

```

```

        orders = Order.objects.all()
        orderl = []
        for order in orders:
            ocf = order.ordercookfoods
            obj = ocf.first()
            ck = obj.cookfoodid.cookid
            if ck == cook:
                orderl.append(order)
        orderl = orderl.order_by('-created')
        return orderl

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

```

Б.2 Текст файла cookfoodview.py

```

from django.contrib.auth.mixins import LoginRequiredMixin
from datetime import datetime
from django.views.generic import ListView
from django.views.generic import UpdateView
from django.views.generic.edit import CreateView
from django.views.generic.edit import DeleteView
from django.views.generic.detail import DetailView
from django.urls import reverse_lazy
from .models import CookFood, Food, Cook, Product,
CookFoodPrice, CookFoodProduct, CookFoodRespond, Client, Cook

class CookFoodView (OrderView):
    template_name = 'homemadecookfood.html'
    model = CookFood

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        now = datetime.today()
        prices = CookFoodPrice.objects.filter(cookfoodid =
self.object.pk, from_date__lte = now, to_date__gte = now)
        context ['price'] = prices.first()
        return context

```

```

class CookFoodAddView(LoginRequiredMixin, CreateView):
    model = CookFood
    fields = ['foodid', 'active', 'recipe']
    template_name = 'cookfoodadd.html'

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

    def get_context_data(self, **kwargs):
        self.foodid = get_object_or_404(Food,
id=self.kwargs['food_id'])
        kwargs['food_id'] = self.foodid
        kwargs['foods'] = Food.objects.all()
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.foodid = get_object_or_404(Food,
id=self.kwargs['food_id'])
        ck = self.request.user
        cook = Cook.objects.get(id = ck)
        form.instance.cookid = cook
        form.instance.rating = 0
        form.instance.since = datetime.today()
        products = str(self.request.POST.get('products'))
        products = products.split(',')
        for prod in products:
            p = Product.objects.get(id = prod)
            cookfp = CookFoodProduct(productid = p,
cookfoodid__id = self.id)
            cookfp.save()
        return super().form_valid(form)

class CookFoodUpdateView(LoginRequiredMixin, UpdateView):
    model = CookFood
    fields = ['foodid', 'active', 'recipe']
    template_name = 'cookupdateupdate.html'

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

    def get_context_data(self, **kwargs):
        self.foodid = get_object_or_404(Food,
id=self.kwargs['food_id'])

```

```

        kwargs['food_id'] = self.foodid
        kwargs['foods'] = Food.objects.all()
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.foodid = get_object_or_404(Food,
id=self.kwargs['food_id'])
        prods = []
        cfs = self.cookfood_cookfoodproducts
        products = str(self.request.POST.get('products'))
        products = products.split(',')
        for prod in products:
            p = Product.objects.get(id = prod)
            prods.append(p)
            cp = CookFoodProduct.objects.get(productid = p,
cookfoodid__id = self.id)
            if not cp:
                newcfp = CookFoodProduct (productid = p,
cookfoodid__id = self.id)
                newcfp.save()
        for cf in cfs:
            if cf.productid not in prods:
                self.cookfood_cookfoodproducts.filter(id =
cf.id).delete()
        return super().form_valid(form)

class CookFoodForCookView(ListView):
    model = CookFood
    template_name = 'cookfoodforcook.html'
    paginate_by = 10

    def get_queryset(self):
        ck = self.request.user
        cook = Cook.objects.get(id = ck.id)
        cookfoods = CookFood.objects.filter(cookid = cook,
active = True)
        cookfoods = cookfoods.order_by('-rating')
        return cookfoods

class CookFoodRespondAddView(LoginRequiredMixin, CreateView):
    model = CookFoodRespond
    fields = ['cookfoodid', 'active', 'recipe']
    template_name = 'cookfoodrespondadd.html'

    def get_success_url(self):

```

```

        return reverse_lazy('cookfoodaddview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        self.cookfoodid = get_object_or_404(CookFood,
id=self.kwargs['cookfood_id'])
        kwargs['cookfood_id'] = self.cookfoodid
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.cookfoodid = get_object_or_404(CookFood,
id=self.kwargs['cookfoodid'])
        cl = self.request.user
        client = Client.objects.get(id = cl)
        form.instance.clientid = client
        form.instance.created = datetime.today()
        cfrs = CookFoodRespond.objects.filter(cookfoodid =
self.cookfoodid)
        total = 0
        numb = cfrs.count()
        for cfr in cfrs:
            total = total + cfr.rating / numb
        cf = self.cookfoodid
        cf.rating = total
        return super().form_valid(form)

class CookFoodRespondsView(ListView):
    model = CookFoodRespond
    template_name = 'cookfoodresponds.html'
    paginate_by = 10

    def get_queryset(self):
        cfid = self.request.GET.get('cookfood')
        responds = CookFoodRespond.objects.filter(cookfoodid =
cfid)

        responds.order_by('-created')
        return responds

    def get_context_data(self, **kwargs):
        context = super(CookFoodRespondsView,
self).get_context_data(**kwargs)
        cfid = self.request.GET.get('cookfood')
        context ['cookfood'] = cfid
        return context

```

```

class CookFoodSaveView(LoginRequiredMixin, CreateView):
    model = SavedCookFood
    fields = ['cookfoodid']
    template_name = 'cookfoodsave.html'

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

    def get_context_data(self, **kwargs):
        self.cookfoodid = get_object_or_404(CookFood,
id=self.kwargs['cookfood_id'])
        kwargs['cookfood_id'] = self.cookfoodid
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.cookfoodid = get_object_or_404(CookFood,
id=self.kwargs['cookfoodid'])
        cl = self.request.user
        client = Client.objects.get(id = cl)
        form.instance.clientid = client
        return super().form_valid(form)

class CookFoodUnsaveView (DeleteView):
    model = SavedCookFood

class CookFoodProductMarkDeleteView (DeleteView):
    model = SavedCookFood

class SavedCookFoodsView(ListView):
    model = SavedCookFood
    template_name = 'savedcookfoods.html'
    paginate_by = 10

    def get_queryset(self):
        ck = self.request.user
        client = Client.objects.get(id = ck.id)
        cookfoods = SavedCookFood.objects.filter(clientid =
cook)

        cookfoods = cookfoods.order_by('-id')
        return cookfoods

class CookFoodSearchView(ListView):
    model = CookFood
    template_name = 'cookfoodsearch.html'

```

```

paginate_by = 10

def get_queryset(self):
    minrating = self.request.GET.get('minrating')
    maxrating = self.request.GET.get('maxrating')
    food = self.request.GET.get('food')
    category = self.request.GET.get('category')
    allms = self.request.GET.get('marks')
    allmarks = []
    allms = allms.split(',')
    for mar in allms:
        m = Product.objects.get(id = mar)
        allmarks.append(m)
    allproducts = self.request.GET.get('products')
    prodds = []
    products = str(self.request.POST.get('products'))
    products = products.split(',')
    for prod in products:
        p = Product.objects.get(id = prod)
        prodds.append(p)
    if not minrating:
        minrating = 0
    if not maxrating:
        maxrating = 10
    foods = CookFood.objects.filter(rating__lte =
minrating, rating__gte = maxrating)
    if category:
        foods = foods.filter(foodid__foodcategoryid__id =
category)

    if food:
        foods = foods.filter(foodid__id = food)
    resultfoods = []
    for f in foods:
        prods = f.cookfood_cookfoodproducts
        prodsres = True
        for p in prods:
            if not (p in prods):
                prodsres = False
        if prodsres:
            for p in prods:
                res = True
                marks = cookfoodproduct_marks
                for mark in allmarks:
                    if not (mark in marks):
                        res = False

```

```

        prodsres = False

        if prodsres:
            resultfoods.append(f)
        resultfoods = resultfoods.order_by('-rating')
        return resultfoods

    def get_context_data(self, **kwargs):
        context = super(CookFoodSearchView,
self).get_context_data(**kwargs)
        minrating = self.request.GET.get('minrating')
        maxrating = self.request.GET.get('maxrating')
        food = self.request.GET.get('food')
        category = self.request.GET.get('category')
        context ['allmarks'] = Mark.objects.all()
        context ['allproducts'] = Product.objects.all()
        context ['allfcategories'] =
Foodcategory.objects.all()
        context ['allcategories'] = Category.objects.all()
        context ['allfoods'] = Food.objects.all()
        allms = self.request.GET.get('marks')
        products = str(self.request.POST.get('products'))
        context ['minrating'] = minrating
        context ['maxrating'] = maxrating
        context ['food'] = food
        context ['category'] = category
        context ['marks'] = allms
        context ['products'] = products
        return context

class CookFoodPriceAddView(LoginRequiredMixin, CreateView):
    model = CookFoodPrice
    fields = ['cookfoodid', 'from_date', 'to_date', 'price',
'about']
    template_name = 'cookfoodpriceadd.html'

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

    def get_context_data(self, **kwargs):
        self.cookfoodid = get_object_or_404(CookFood,
id=self.kwargs['cookfood_id'])
        kwargs['cookfood_id'] = self.cookfoodid
        return super().get_context_data(**kwargs)

```

```

    def form_valid(self, form):
        self.cookfoodid = get_object_or_404(CookFood,
id=self.kwargs['cookfoodid'])
        return super().form_valid(form)

class CookFoodPriceUpdateView(LoginRequiredMixin, UpdateView):
    model = CookFood
    fields = ['cookfoodid', 'from_date', 'to_date', 'price',
'about']
    template_name = 'cookfoodpriceupdate.html'

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

    def get_context_data(self, **kwargs):
        self.cookfoodid = get_object_or_404(CookFood,
id=self.kwargs['cookfood_id'])
        kwargs['cookfood_id'] = self.cookfoodid
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.cookfoodid = get_object_or_404(CookFood,
id=self.kwargs['cookfoodid'])
        return super().form_valid(form)

class CookFoodProductMarkAddView(LoginRequiredMixin, CreateView):
    model = CookFoodProductMark
    fields = ['cookfoodproductid', 'markid']
    template_name = 'cookfoodproductmarkadd.html'

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

    def get_context_data(self, **kwargs):
        self.cookfoodproductid =
get_object_or_404(CookFoodProduct,
id=self.kwargs['cookfoodproduct_id'])
        kwargs['cookfoodproduct_id'] = self.cookfoodproductid
        self.markid = get_object_or_404(Mark,
id=self.kwargs['mark_id'])
        kwargs['mark_id'] = self.markid
        kwargs['marks'] = Mark.objects.all()
        return super().get_context_data(**kwargs)

```

```

    def form_valid(self, form):
        self.markid = get_object_or_404(Mark,
id=self.kwargs['mark_id'])
        self.cookfoodproductid =
get_object_or_404(CookFoodProduct,
id=self.kwargs['cookfoodproduct_id'])
        return super().form_valid(form)

```

Б.3 Текст файла models.py

```

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

STATES = [
    ('A', 'передзамовлення'),
    ('B', 'підтверджене'),
    ('C', 'готове'),
    ('D', 'відправлене'),
    ('E', 'отримане'),
    ('F', 'скасоване'),
]

class Cook (AbstractUser):
    address = models.TextField()
    rating = models.DecimalField(max_digits = 2, decimal_places
= 1, default = 0)
    since = models.DateTimeField()
    phone = models.CharField(max_length=10)

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

class Client (AbstractUser):
    address = models.TextField()
    phone = models.CharField(max_length=10)

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

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

```

```

class Food (models.Model):
    foodcategoryid = models.ForeignKey(FoodCategory,
related_name='categoryfoods', on_delete=models.CASCADE)
    about = models.TextField()
    named = models.CharField(max_length=50)

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

    def __str__(self):
        return self.named

class CookFood (models.Model):
    cookid = models.ForeignKey(Cook,
related_name='cook_cookfoods', on_delete=models.CASCADE)
    foodid = models.ForeignKey(Food,
related_name='food_cookfoods', on_delete=models.CASCADE)
    since = models.DateTimeField()
    active = models.BooleanField()
    recipe = models.TextField()
    alternativenamed = models.CharField(max_length=100)
    rating = models.DecimalField(max_digits = 2, decimal_places
= 1, default = 0)

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

class Foodcategory (models.Model):
    about = models.TextField()
    named = models.CharField(max_length=30)

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

    def __str__(self):
        return self.named

class Order (models.Model):
    clientid = models.ForeignKey(Client,
related_name='clientorders', on_delete=models.CASCADE)
    deliveryid = models.ForeignKey(Delivery,
related_name='orderdeliveries', on_delete=models.CASCADE)
    created = models.DateTimeField()
    for_date = models.DateTimeField()

```

```

        readytime = models.DateTimeField()
        comment = models.CharField(max_length=255)
        pay = models.DecimalField(max_digits = 10, decimal_places =
1, default = 0)
        state = models.CharField(
            max_length=1,
            choices = STATES,
            default = 'A',
        )

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

class OrderCookFood (models.Model):
    orderid = models.ForeignKey(Order,
related_name='ordercookfoods', on_delete=models.CASCADE)
    cookfoodid = models.ForeignKey(CookFood,
related_name='cookfoodordercookfoods', on_delete=models.CASCADE)
    amount = models.IntegerField()
    comment = models.CharField(max_length=255)
    active = models.BooleanField()
    pay = models.DecimalField(max_digits = 10, decimal_places =
1, default = 0)

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

    def save(self, *args, **kwargs):
        price = CookFoodPrice.objects.filter(cookfoodid =
self.cookfoodid, from_date__lte = self.orderid.for_date, to_date__gte
= self.orderid.for_date)
        self.pay = price.pay
        super(OrderCookFood, self).save(*args, **kwargs)

class Category (models.Model):
    named = models.CharField(max_length=30)

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

    def __str__(self):
        return self.named

```

```

class Product (models.Model):
    named = models.CharField(max_length=30)
    categoryid = models.ForeignKey(Category,
related_name='categoryproducts', on_delete=models.CASCADE)

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

    def __str__(self):
        return self.named

class Mark (models.Model):
    named = models.CharField(max_length=30)

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

    def __str__(self):
        return self.named

class CookFoodProduct (models.Model):
    cookfoodid = models.ForeignKey(CookFood,
related_name='cookfood_cookfoodproducts', on_delete=models.CASCADE)
    productid = models.ForeignKey(Product,
related_name='product_cookfoodproducts', on_delete=models.CASCADE)

class CookFoodProductMark (models.Model):
    cookfoodproductid = models.ForeignKey(CookFoodProduct,
related_name='cookfoodproduct_marks', on_delete=models.CASCADE)
    markid = models.ForeignKey(Mark,
related_name='mark_cookfoodproducts', on_delete=models.CASCADE)

class CookFoodPrice (models.Model):
    cookfoodid = models.ForeignKey(CookFood,
related_name='cookfood_cookfoodprices', on_delete=models.CASCADE)
    from_date = models.DateTimeField()
    to_date = models.DateTimeField()
    about = models.TextField()
    pay = models.DecimalField(max_digits = 10, decimal_places =
1, default = 0)

class CookFoodRespond (models.Model):
    cookfoodid = models.ForeignKey(CookFood,
related_name='cookfood_cookfoodresponds', on_delete=models.CASCADE)

```

```

        clientid = models.ForeignKey(Client,
related_name='clientcookfoodresponses', on_delete=models.CASCADE)
        created = models.DateTimeField()
        respond = models.TextField()
        rating = models.DecimalField(max_digits = 2, decimal_places
= 1, default = 0)

class CookRespond (models.Model):
    cookid = models.ForeignKey(Cook,
related_name='cook_cookresponds', on_delete=models.CASCADE)
    clientid = models.ForeignKey(CookFood,
related_name='clientcookresponds', on_delete=models.CASCADE)
    created = models.DateTimeField()
    respond = models.TextField()
    rating = models.DecimalField(max_digits = 2, decimal_places
= 1, default = 0)

class SavedCookFood (models.Model):
    cookfoodid = models.ForeignKey(CookFood,
related_name='cookfoodsavedcookfoods', on_delete=models.CASCADE)
    clientid = models.ForeignKey(CookFood,
related_name='clientsavedcooks', on_delete=models.CASCADE)

class Message (models.Model):
    cookid = models.ForeignKey(Cook,
related_name='cookmessages', on_delete=models.CASCADE)
    clientid = models.ForeignKey(CookFood,
related_name='clientmessages', on_delete=models.CASCADE)
    created = models.DateTimeField()
    about = models.TextField()
    direction = models.BooleanField()

    def __str__(self):
        return self.about

class CookDelivery (models.Model):
    cookid = models.ForeignKey(Cook,
related_name='cook_cookdeliveries', on_delete=models.CASCADE)
    deliveryid = models.ForeignKey(Delivery,
related_name='delivery_cookdeliveries', on_delete=models.CASCADE)
    price = models.TextField()

class Delivery (models.Model):
    named = models.CharField(max_length=30)

```

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

def __str__(self):
    return self.named

class SavedCook (models.Model):
    cookid = models.ForeignKey(Cook,
related_name='cook_cooksavedcooks', on_delete=models.CASCADE)
    clientid = models.ForeignKey(Client,
related_name='client_savedcooks', on_delete=models.CASCADE)
```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ЗАМОВЛЕННЯ ЇЖІ ДОМАШНЬОГО ПРИГОТУВАННЯ

Дипломна кваліфікаційна робота

Виконав: студент групи КНТ-138

В.О. Тарусов

Керівник:

Т.В. Голуб

2022

Рисунок В.1 – Слайд № 1

Об'єкт, предмет, мета роботи

Об'єктом роботи є процес замовлення їжі домашнього приготування.

Предметом роботи є програмне забезпечення замовлення їжі домашнього приготування.

Мета роботи – розробити програмне забезпечення замовлення їжі домашнього приготування для встановлення взаємодії між кухарями і клієнтами.

Рисунок В.2 – Слайд № 2

Завдання роботи

- ✓ аналіз предметної області;
- ✓ визначення вимог до розробки;
- ✓ проєктування програмного забезпечення замовлення їжі домашнього приготування;
- ✓ програмна реалізація програмного забезпечення замовлення їжі домашнього приготування.

3

Рисунок В.3 – Слайд № 3

Аналіз предметної області

Критерій	HomeMade	Savorly	Моя програма
Доступність в Україні	Ні	Ні	Так
Пошук страв домашнього приготування	Так	Так	Так
Визначення фільтрів страв	Обмежене	Обмежене	Розширюване
Замовлення страв	Так	Так	Так
Перелік продуктів у страві	Так	Так	Так
Швидкий доступ до обраних страв	Так	Так	Так
Швидкий доступ до обраних кухарів	Ні	Ні	Так
Вибір служби доставки	Так	Ні	Так

4

Рисунок В.4 – Слайд № 4

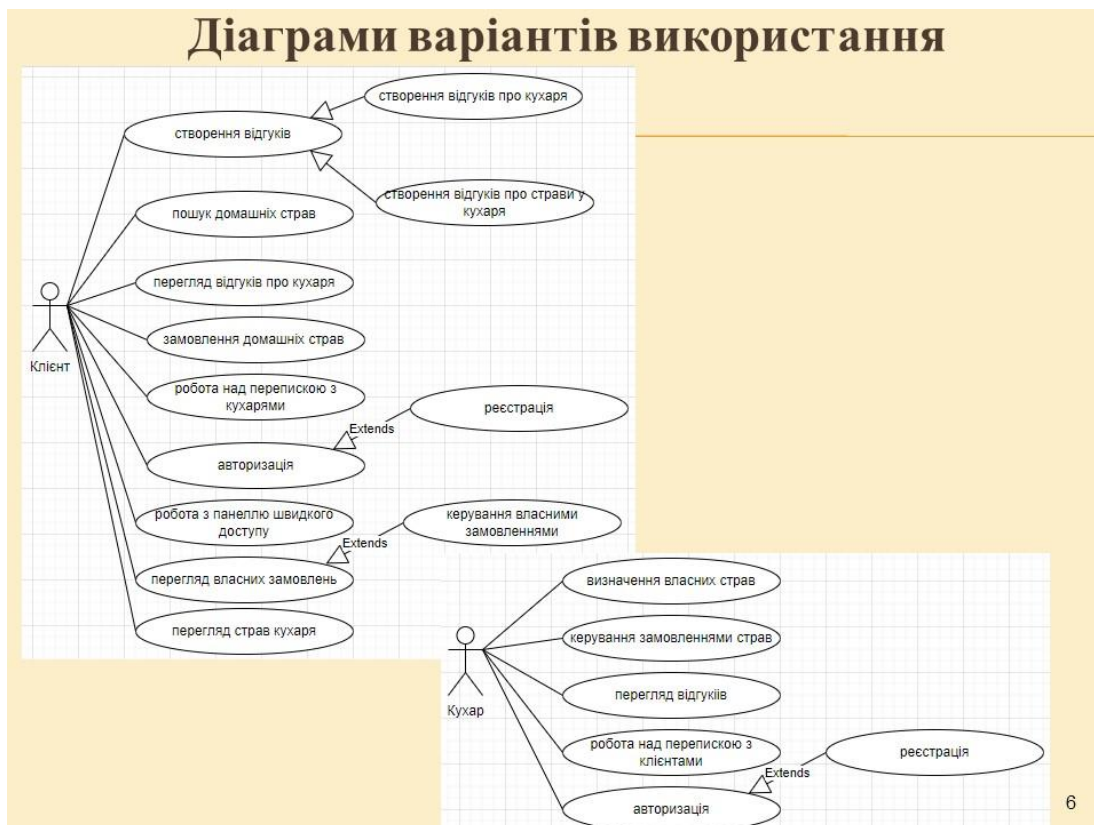
Вибір засобів розробки

Критерій	Python	Java
Динамічність розробки	Вища	Нижча
Читабельність коду	Вища	Нижча
Фреймворки	Django, Flask	Spring, Blade
Стабільність	Однакова	Однакова

5

Рисунок В.5 – Слайд № 5

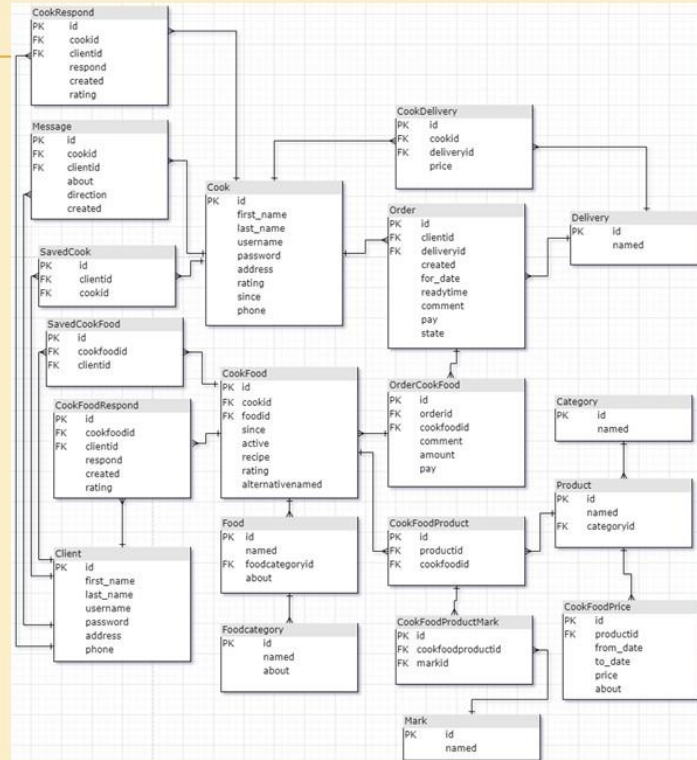
Діаграми варіантів використання



6

Рисунок В.6 – Слайд № 6

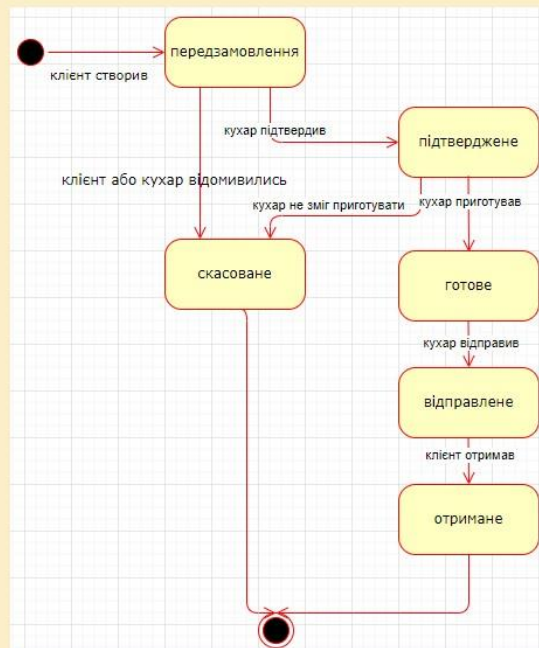
Схема бази даних



7

Рисунок В.7 – Слайд № 7

Діаграма станів замовлення їжі домашнього приготування



8

Рисунок В.8 – Слайд № 8

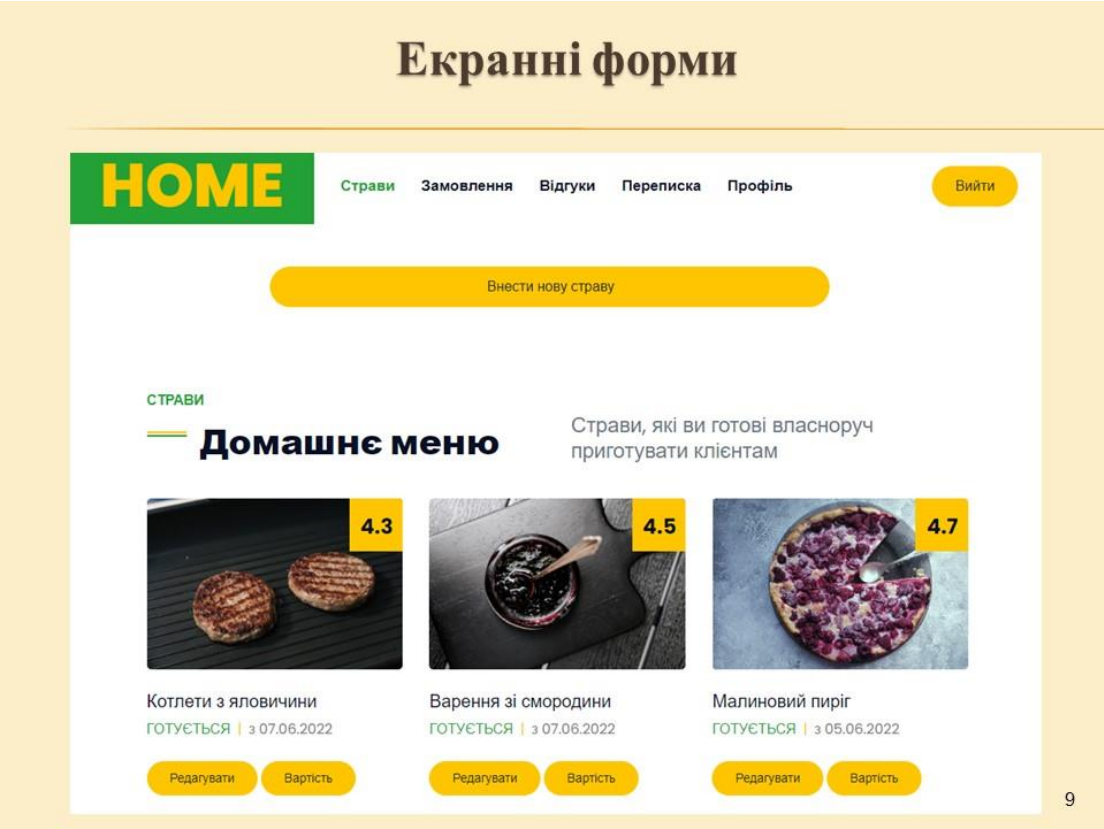


Рисунок В.9 – Слайд № 9

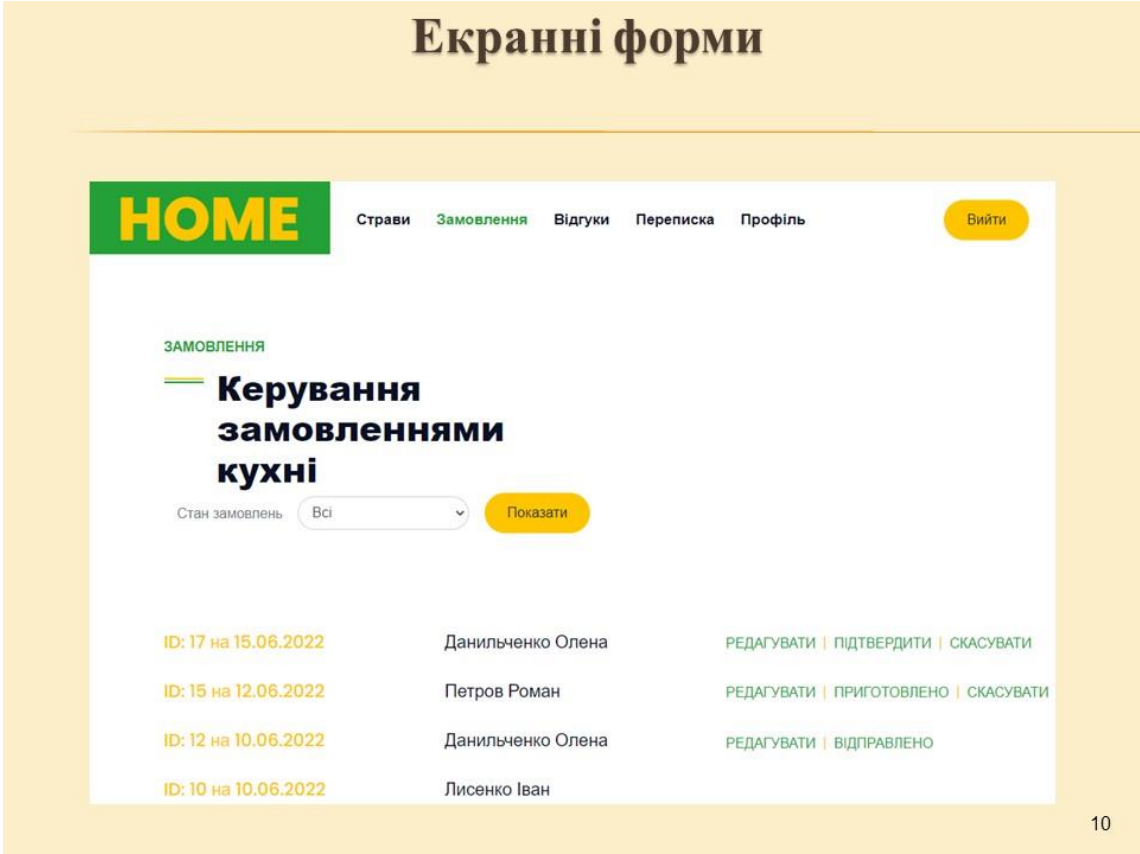


Рисунок В.10 – Слайд № 10

Екранні форми

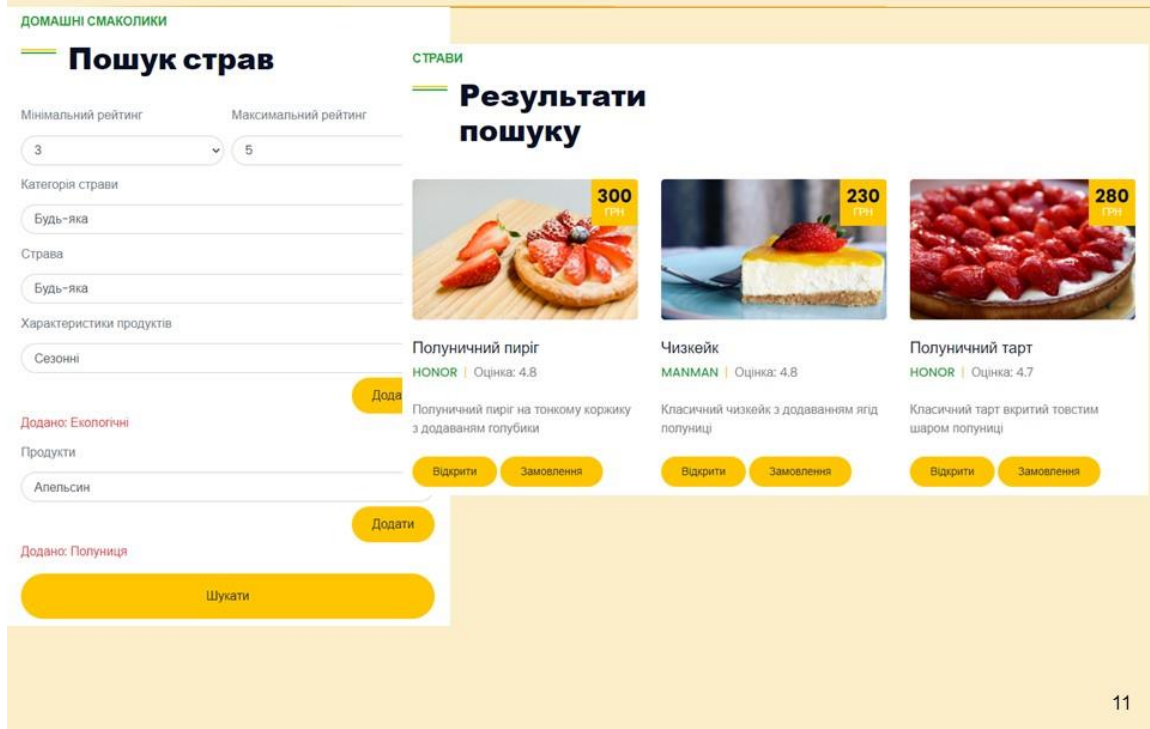


Рисунок В.11 – Слайд № 11

Висновки

Програма реалізована як вебзастосунок, що надає можливості визначення страв, які готує кухар, створення відгуків про страви у кухаря і про кухаря, замовлення домашніх страв у кухаря, керування замовленнями страв, пошук домашніх страв, перегляд замовлень клієнтом, перегляд відгуків про кухаря, збереження страв кухаря у власний перелік для швидкого доступу, а даних кухаря у власний перелік для швидкого доступу, надсилання повідомлень, відтворення переписки, перегляду страв кухаря.

Розроблена програма дозволяє підтримувати роботу бажаючих готувати вдома для інших людей шляхом організації замовлень таких страв, керування цими замовленнями, підтримувати клієнтів, які бажають визначити те, які страви прийдуть їм до смаку і знайти серед цього різноманіття саме те, що дозволить харчуватися домашньою і до того ж можливо здоровою їжею.

Рисунок В.12 – Слайд № 12