Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерних наук_____
(повна назва)
Кафедра_____Програмної інженерії_____
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти - другий (магістерський)

Дослідження задач лінійного програмування для побудови бізнес-логіки системи підтримки мережі онлайн-закладів харчування
(тема)

Виконав: студент 2 курсу, групи ІПЗм-18-4___
_____Жаренков К.К._____
(прізвище, ініціали)

спеціальності 121- Інженерія програмного забезпечення
(код і повна назва спеціальності)

Освітньо-наукової програми_____
(тип програми)

Інженерія програмного забезпечення___
(повна назва освітньої програми)

Керівник _____доцент, к.т.н. Вечур О.В._____
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2020р.

Харківський національний університет радіоелектроніки

Факультет_Комп'ютерних наук_____
Кафедра_Програмної інженерії_____
Рівень вищої освіти - другий (магістерський)

Спеціальність _121-Інженерія програмного забезпечення_____
(код і повна назва)
Тип програми освітньо-наукова програма

Освітня програма_Інженерія програмного забезпечення _____

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 20 ___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові _ Жаренкову Кирилу Костянтиновичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи _Дослідження задач лінійного програмування для побудови бізнес-логіки системи підтримки мережі онлайн-закладів харчування_____
затверджена наказом університету від "_27_" __03___2020 р № ___473 Ст__
заповнюється вручну після отримання наказу
2. Термін подання студентом роботи до екзаменаційної комісії
15 травня 2020 р.
3. Вихідні дані до роботи *електронні ресурси за обраною тематикою, мінімальні вимоги до функціональності програми, загальні вимоги до архітектури системи*_____
4. Перелік питань, що потрібно опрацювати в роботі *аналіз предметної області і постановка задачі, аналіз вимог до програмної системи, UML-моделювання предметної області, дослідження задач лінійного програмування, розробка математичних моделей оптимізаційних розподільчих задач та задач про призначення, розробка схеми бази даних, розробка алгоритмів вирішення оптимізаційних задач лінійного програмування, програмна реалізація системи, тестування.*

5 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	доцент, к.т.н. Вечур О.В.		15.05.200

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Аналіз проблемної області та постановка задачі	27.01.20-08.02.20	виконано
2	Формування вимог до програмної системи	08.02.20-11.02.20	виконано
3	Аналіз та UML-моделювання предметної області	08.02.20-20.02.20	виконано
4	Дослідження задач лінійного програмування	15.02.20-29.02.20	виконано
5	Розробка математичних моделей	25.02.20-10.03.20	виконано
6	Розробка схеми бази даних	10.03.20-15.03.20	виконано
7	Розробка алгоритмів вирішення оптимізаційних задач лінійного програмування	15.03.20-30.03.20	виконано
8	Програмна реалізація та тестування системи	16.03.20-01.04.20	виконано
9	Підготовка пояснювальної записки	15.03.20-8.05.20	виконано
10	Підготовка презентації та доповіді	10.05.20-12.05.20	виконано
11	Перевірка на плагіат	08.05.20	виконано
12	Попередній захист	15.05.20	виконано
13	Нормоконтроль, рецензування	10.05.20	виконано
14	Занесення диплома в електронний архів	12.05.20	виконано
15	Допуск до захисту у зав. кафедри	15.05.20	виконано

Дата видачі завдання __27___січня_____2020 р.

Студент _____ Жаренков К.К._____
(підпис)

Керівник роботи _____ доцент, к.т.н. Вечур О.В___
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Атестаційна робота магістра містить: 100 с., 24 рис., 5 табл., 27 джер.

БАЗА ДАНИХ, БІЗНЕС-ЛОГІКА, ЗАКЛАД ХАРЧУВАННЯ, ЛІНІЙНЕ ПРОГРАМУВАННЯ, ОПТИМІЗАЦІЙНА ЗАДАЧА, МОДЕЛЮВАННЯ, ТАБЛИЦЯ, ANDROID, JSON, POSTGREEDB.

Об’єктом дослідження є задачі лінійного програмування для моделювання бізнес-логіки роботи мережі онлайн-закладів харчування.

Метою роботи є підвищення ефективності бізнес-процесів в системі підтримки мережі онлайн-закладів харчування.

Методи розробки базуються на таких технологіях, як Android SDK, Java, Spring, Retrofit.

В результаті роботи було досліджено задачі лінійного програмування та проведено моделювання бізнес-процесів розподілу складних замовлень між закладами харчування та призначення водіїв на доставку замовлень, проведено аналіз та моделювання предметної області, розроблено схему бази даних та програмно реалізовано основні функції роботи з базою даних.

DATABASE, BUSINESS LOGIC, POWER ESTABLISHMENT, LINEAR PROGRAMMING, OPTIMIZATION PROBLEM, MODELING, TABLE, ANDROID, JSON, POSTGREEDB/

The object of the study is the problem of linear programming to model the business logic of the network of online restaurants.

The aim of the work is to increase the efficiency of business processes in the support system of the network of online restaurants.

Development methods are based on such technologies as Android SDK, Java, Spring, Retrofit.

As a result, the problems of linear programming were investigated and business processes of distribution of complex orders between catering establishments and

appointment of drivers for delivery of orders were analyzed, analysis and modeling of subject area was performed, database schema was developed and basic database functions were implemented.

ЗМІСТ

ВСТУП

У сучасному світі достатньо розповсюджена стала доставка їжі, як частина життя нинішньої людини. Важко уявити сучасне життя без швидких та сучасних сервісів доставки їжі та напоїв. Мільйони людей по всьому світу замовляють їжу додому та до офісів. Тому дуже важливо надати якісний сервіс, який буде заощаджувати час та гроші клієнтів. Вже існує багато проектів, які спеціалізуються на доставці їжі та напоїв, але і під час здавалось би налагодженої роботи можуть виникати складності, що пов'язані з бізнес-логікою, наприклад, обслуговування великих замовленнями та інше.

На жаль, дуже складно організувати усю логістику з доставки їжі закладу самостійно, треба витрачати багато грошей на рекламу, кур'єрів та систему для доставки їжі. У зв'язку з цим, сьогодні дуже розповсюдженими стали компанії з доставки їди, таки як Glovo та Uber Eats, кожен з яких пропонує узяти на себе усі складності із доставкою, а закладу залишається тільки приготувати вчасно їжу. Це дозволяє зменшити витрати на доставку, обслуговування машин, створення чи покупки спеціальної системи для доставки.

Отже, постає питання про створенні програмної системи для онлайн-замовлення їжі, яка буде підтримувати бізнес-логіку мережі закладів харчування, таких як розподіл навантаження між кафе, підтримки доставки замовлень та інше.

Важливим завданням для створення такої системи є розробка моделей та алгоритмів з оптимального розподілу навантажень на приготування великих замовлень, з оптимального призначення водіїв на доставку, тощо. Такого роду завдання можна вирішити на базі оптимізаційних задач лінійного програмування (ЗЛП). В роботі проведено дослідження ЗЛП з метою моделювання відповідної бізнес-логіки.

Було розроблено оптимізаційні модель задач з розподілу великого замовлення між кафе для приготування та задач про призначення водіїв на

доставку, а також було розроблено алгоритми вирішення таких завдань і програмно реалізувано їх в складі програмної системи.

У результаті розроблено програмну систему підтримки мережі онлайн закладів харчування, яка виконує всі функції, які необхідні для швидкої і зручної доставки їжі та напоїв будь-яких об’ємів. Система складається з декількох частин: серверу бази даних (БД), серверу додатку, веб- та мобільного додатку. Сервер додатку виконує головну функцію – вирішує оптимізаційні задачі. Мобільний додаток надає змогу водіям, які користуючись геолокаційними сервісами свого смартфону, можуть заробляти кошти на доставці, реєструючись в системі як водії-фрілансери. Веб-додаток дозволяє кліектові замовити доставку їжі, а адміністратору проводити роботу з різними об’єктами даних, що присутні в системі: заклад харчування, страва, продукт, інгредієнт, доставка та інше.

За результатами роботи розроблено презентацію (див. додаток А). Частина коду наведена в додатку Б. Тези доповіді на Молодіжному форумі наведено в додатку В. Електронні матеріали за роботою наведено на диску (див. додаток Г).

1 АНАЛІЗ ПРОБЛЕМНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблемної області створення програмних засобів для підтримки закладів харчування

Популярність послуги з доставки їжі додому або в офіс росте з кожним днем. Це пов'язано із застосуванням нових видів сервісів [1], що дозволяють зручно оформити замовлення, розширенням бази користувачів інтернету, яким подобається швидка доставка їжі.

Кар'єра і успіх сьогодні безпосередньо залежать від того, скільки сил і часу віддається роботі, тому сучасна людина вважає за краще користуватися послугами кафе в обідню перерву, із задоволенням проводить час в улюбленому ресторані ввечері. Але не завжди є можливість замовити заздалегідь столик, вийти з дому для того, щоб повечеряти. Замовлення доставки будь-яких ресторанних страв дозволяє знайти вихід з положення.

Великі мережі ресторанів або закладів харчування обов'язково мають свої сторінки в інтернеті, де можна не тільки переглянути все меню, а й оформити замовлення до дому або в офіс. Але не всі заклади можуть створити свій якісний сайт, який стає сьогодні головним джерелом поповнення клієнтської бази. Тут, на допомогу приходять агрегатори доставки [2], які, співпрацюючи практично з усіма ресторанами і кафе в великих містах, збирають на сторінках свого сайту меню, пропонуючи великий вибір страв тим, хто хоче швидко організувати смачний обід або вечерю.

Великі компанії, такі як Glovo [2], стали сьогодні особливо затребуваними, завдяки чітко організованій роботі, використанню сучасних методів і грамотної маркетингової політики. Тисячі зареєстрованих користувачів, сотні ресторанів і кафе, величезна кількість прийнятих замовлень щодня, стають показником успішної роботи. Серед усіх видів доставки, саме сервіси, що працюють в області громадського харчування, є особливо

популярними, затребуваними і зручними, завдяки використанню сайтів і мобільних додатків.

Мережа з кількох закладів по всьому місту - це практично готова база для організації сервісу доставки їжі. Для цього виду бізнесу вкрай важливо налагодити добре працюючу кур'єрську мережу, щоб замовлення приходили вчасно і в належній якості. Практика показує, що експрес-доставка їжі - найскладніший сегмент не тільки в ресторанному бізнесі, але і в логістиці [3].

Доставка з мережевих ресторанів організована по більш складним принципам, ніж з окремого закладу. Оператор колл-центру повинен не тільки прийняти і оформити замовлення, але і вирішити, з якої точки буде зручніше доставити їжу за адресою, як краще завантажити кур'єрів, щоб вони обслужили найбільшу кількість замовлень за один виїзд. Під час роботи бізнес-логіки обслуговування клієнтів можливі помилки:

а) невміння планувати пікові навантаження;

б) брак кур'єрів;

в) перевантаження окремих ресторанів - занадто багато замовлень;

г) погана робота кур'єрів.

Наприклад, при розподілі замовлень з колл-центру по закладах харчування, де їх будуть виконувати, можна зіткнутися з перевантаженням конкретної точки. Це трапляється, якщо з одного району міста одночасно надходить багато замовлень, а оператор не враховує поточне завантаження закладу харчування, що працює на цей район.

Отже, для вирішення проблем з реалізацією бізнес-логіки [2] таких виробничих процесів як доставка їжі необхідно застосовувати науковий підхід. Наприклад, проблему доставки великих замовлень їжі можна вирішити за допомогою ефективного розподілу водіїв між замовленнями, що здатно знизити вартість доставки, а отже і поліпшити сервіс загалом.

Науковий напрямок, який займається проблемами організаційного керування, має назву теорія прийняття рішень (ТПР) [4, 5], що дозволяє знайти оптимальне вирішення тієї чи іншої проблеми. Моделям та методам прийняття

оптимальних рішень (ефективних рішень) в даний час приділяється дуже велика увага. Методи пошуку оптимальних рішень розглядають в розділах ТПР, а саме в теорії дослідження операцій, яка вирішую так звані задачі математичного програмування (ЗМП) [6] .

Задачі математичного програмування вирішуються на замовлення керівників: знайти найкраще (оптимальне), єдино вірне і науково обгрунтоване рішення. Отримавши таке замовлення аналітик досліджує систему, зовнішнє середовище і намагається знайти адекватну модель.

Використовувані моделі носять об'єктивний характер. Побудова моделей розглядається в рамках дослідження операцій як засіб відображення об'єктивно існуючої реальності. Існує об'єктивний критерій успіхів в застосуванні методів дослідження операцій. Якщо проблема, яка потребує вирішення, ясна, критерій визначено, то аналітичний метод відразу показує, наскільки нове рішення краще старого.

Побудова моделей є основою системного підходу до розв'язання задач організаційного управління і допомагає привести складні і невизначені чинники, пов'язані з проблемою ПР, до логічної схеми. Під час побудови математичної моделі оптимізаційної ЗМП необхідно:

- визначити для яких величин має бути побудована модель, іншими словами, як ідентифікувати змінні (шукані величини) даної задачі;

- визначити обмеження, які мають бути накладені на змінні, щоб виконувалися умови, що характеризують модельовану систему;

- визначити у чому полягає ціль, для досягнення якої із усіх припустимих значень змінних необхідно обрати саме ті, що відповідатимуть оптимальному розв’язанню задачі.

На сьогодні існує наступна класифікація ЗМП [6]:

- якщо всі функції в моделі лінійні, то відповідна задача є задачею лінійного програмування (ЗЛП) [7];

- задачі цілочисельного програмування – тут невідомі набувають тільки цілочисельних значень;

- якщо хоча одна із функцій моделі нелінійна – задачі нелінійного програмування, до них відносять задачі опуклого програмування, квадратичного програмування, дискретного програмування, параметричного програмування;

- задачі дрібно-лінійного програмування - цільова функція являє собою відношення двох лінійних функцій, а функції, що визначають область можливих змін невідомих, також є лінійні;

- задачі динамічного програмування - задача, процес знаходження рішення якої є багатоетапним (багатокроковим) та інші.

Бізнес-логіка роботи закладів харчування може бути промодельована за допомогою математичних моделей з лінійними залежностями, отже для наукового підходу з вирішення відповідних проблем підійдуть задачі лінійного програмування, або розподільчі задачі [6, 7].

Загальне формулювання ЗЛП таке [7]: відомі об’єми наявності кожного і-го типу ресурсів bi, об’єми необхідних ресурсів для виконання кожного із j-х типів робіт aj, питомі витрати і-го типу ресурсу на виконання j-го типу робіт Cij. Потрібно знайти такий розподіл ресурсів за роботами, щоб мінімізувалися сумарні витрати на виконання усіх робіт, або максимізувався одержуваний у підсумку загальний прибуток.

Отже, в загальній оптимізаційній моделі [7] використовують:

– параметри Cij - витрати або прибуток, щодо виділення однієї одиниці ресурсу Ri до роботи Jj;

– шукані невідомі Xij, зазвичай це об’єм і-го ресурсу, необхідного для виконання j-ї роботи.

Загальні витрати (або прибуток) на використання і-го ресурсу для виконання j-ї роботи дорівнюють Xij*Cij, а це лінійна розподільна задача [6, 7] з цільовою функцією й системою обмежень, які наведено нижче:

$$\sum_{j=1}^{n}\sum_{i=1}^{m} C_{ij} * X_{ij} \rightarrow extremum,$$

$$\sum_{j=1}^{n} X_{ij} \le b_i, \ \ \forall i = \overline{1,m},$$

$$\sum_{i=1}^{m} X_{ij} \ge a_j, \ \ \forall j = \overline{1,n},$$

$$X_{ij} \ge 0.$$

В залежності від обмежень в моделі ЗЛП існує відповідна класифікація ЗЛП, які можуть використовуватися для вирішення окремих бізнес-задач, наприклад:

- задачі про призначення;
- задачі про суміші;
- -задачі про розкрій або розпіл;
- транспортні здачі та інші.

Відповідно, для вирішення різних ЗЛП існують свої методи їх вирішення:

- симплекс-метод;
- різни види переборних методів;
- жадібні алгоритми [8] та інше.

Для використання зазначеного наукового підходу інформацію з розроблених моделей необхідно зберігати в інформаційних системах [9]. Цінність будь-якої інформаційної системи полягає у бізнес-даних, які вона має. Тому дуже важливим етапом створення програмних систем є вибір бази даних (БД). Існують різні підходи до організації даних [10]: ієрархічний, реляційний, об’єктно-орієнтований, NoSQL-підхід та інші. Всі вони мають свої недоліки та переваги, однак для побудови великих програмних комплексів використовують реляційні бази даних. Адже вони мають чітке математичне обґрунтування до збереження даних. Проте існують ситуації, в яких небажано використовувати реляційний підхід, адже він може знизити швидкість виконання запитів. В таких ситуаціях слід звернути увагу на NoSQL системи, які мають здатність швидко реагувати на зміну даних. Слід звернути увагу на те, що NoSQL підхід

є зручним та ефективним для зберігання простих даних, а реляційний – для складних..

Для зберігання та обробки даних щодо створення замовлень онлайн підходить реляційна база даних (РБД) . РБД - база даних, побудована на основі реляційної моделі. Кожна таблиця РБД представляється як сукупність рядків і стовпців, де рядки відповідають екземпляру об'єкта, конкретної події або явища, а стовпці - атрибутам (ознаками, характеристиками, параметрами) об'єкта, події, явища. РБД надають більш простий доступ до звітів, які оперативно створюються (зазвичай через SQL [11]) і забезпечують підвищену надійність і цілісність даних завдяки відсутності надлишкової інформації.

Проектування РБД передбачає виконання наступних етапів:

- аналіз предметної області (ПО);

- концептуальне моделювання ПО;

- логічне моделювання, яке супроводжується виконанням нормалізації отриманої логічної моделі;

- побудова фізичної моделі БД.

Концептуальне (інфологічне) проектування [10] – побудова семантичної (змістовної) моделі предметної області. Така модель створюється без орієнтації на якусь конкретну СКБД і модель даних. Найчастіше концептуальна модель бази даних включає в себе:

– опис інформаційних об'єктів, або понять предметної області і зв'язків між ними;

– опис обмежень цілісності, тобто вимог до допустимих значень даних і до зв'язків між ними та інше.

Логічне проектування – створення схеми бази даних на основі конкретної моделі даних, наприклад, реляційної моделі даних. Реляційна модель даних повинна бути нормалізована. Нормалізація відношень [10, 12] – покроковий процес розділення (декомпозиції) початкових відношень БД на простіші. Кроки цього процесу переводять схему відношення БД в послідовні нормальні форми.

Кожна наступна форма володіє кращими властивостями ніж попередня. Як правило [10, 12] БД нормалізують до 3 нормальної форми, тому буде доречним згадати, які обмеження мають перші три нормальні форми: 1NF, 2NF і 3NF.

Під час розробки фізичної моделі бази даних слід пам’ятати про те, що специфіка конкретної СКБД може включати в себе обмеження на іменування об'єктів бази даних, обмеження на підтримувані типи даних і т.п. Крім того, специфіка конкретної СУБД при фізичному проектуванні включає вибір рішень, пов'язаних з фізичним середовищем зберігання даних [10] (вибір методів управління дисковою пам'яттю, поділ БД по файлах і пристроїв, методів доступу до даних), створення індексів і т.д.

Неабияк важливим чинником успішного програмного продукту є його архітектура [9, 13]. Перш за все вона повинна бути маштабованою та гнучкою. Це означає, що з часом дуже легко вносити якісь нові зміни до системи. Для реалізації програмної системи з доставки замовлень слід зупинитися на архітектурі клієнт-сервер [9], що дозволяє мати всю важливу логіку безпосередньо в одному місці – на сервері. Це неабияк спрощує внесення майбутніх змін, наприклад, до алгоритму пошуку водіїв, адже потрібно тільки оновити в одному місці. Клієнтам – мобільним додаткам водіїв – зовсім не потрібно знати як працює алгоритм, які сервіси він використовує, яка база даних зберігає дані, тощо.

Для проектування системи можна використати архітектурний шаблон клієнт-сервер, який вважається домінуючою концепцією у створенні розподілених мережних застосунків і передбачає взаємодію та обмін даними між ними. Клієнт-серверна архітектура - це така архітектура, в якій завдання, поставлені перед системою умовно діляться між двома підсистемами: сервером та клієнтом.

Для вирішення подібних бізнес-задач необхідно залучити інформаційні технології. Вони здатні автоматизувати весь цикл обробки замовлення: від вибору бажаних страв до кінцевої доставки усього замовлення. Тому, перш за все необхідно обрати технології, які здатні вирішити поставлену задачу

максимально швидко та ефективно [14, 15]. При виборі мов програмування слід зупинитися на тих, які дозволяють більше часу приділяти вирішенню бізнес-задачі.

1.2 Аналіз аналогів

На сьогоднішній день існує досить багато різноманітних сервісів із доставки їжі. В кожному місті існують свої подібні сервіси, популярність яких відрізняється в залежності від міста. Більшість з них вирішує проблему із доставкою делегуючи це завдання своїм кур’єром, через що закладам харчування не потрібно мати своїх робітник із доставки. Зазвичай ці сервіси працюють як агрегатори, в якому можна побачити список закладів та їх меню.

Glovo - іспанський стартап, сервіс доставки їжі через мобільний додаток (див. рис. 1.1). Станом на 2019 компанія оперує у більш ніж 170 містах, 24 країнах [2]. З сервісом співпрацює більше 25 000 кур'єрів.

Замовити їжу можна через додаток, воно є на iOS і Android. Також замовлення можна здійснювати через сайт - головне, зареєструватися в системі. Далі через Glovo можна замовити що завгодно: готову їжу з ресторанів, закусочних і барів, медикаменти з аптек, продукти з супермаркетів і взагалі будь-яку доставку. Серед основних функцій платформи: визначення місця розташування кур'єра, що дозволяє користувачам стежити за пересуванням їх покупки в реальному часі, а також можливість зв'язатися з найближчим кур'єром, щоб таким чином оптимізувати час доставки.

«ЕКІПАЖ СЕРВІС» – сервіс доставки їжі, який працює в 25 містах України. Замовлення можна зробити з більш ніж 800 закладів української, європейської, китайської кухні, в тому числі доступні і вегетаріанські страви. Працює сервіс цілодобово, а замовлення привезуть протягом години. Є веб-сайт, мобільні додатки для Android та iOS.

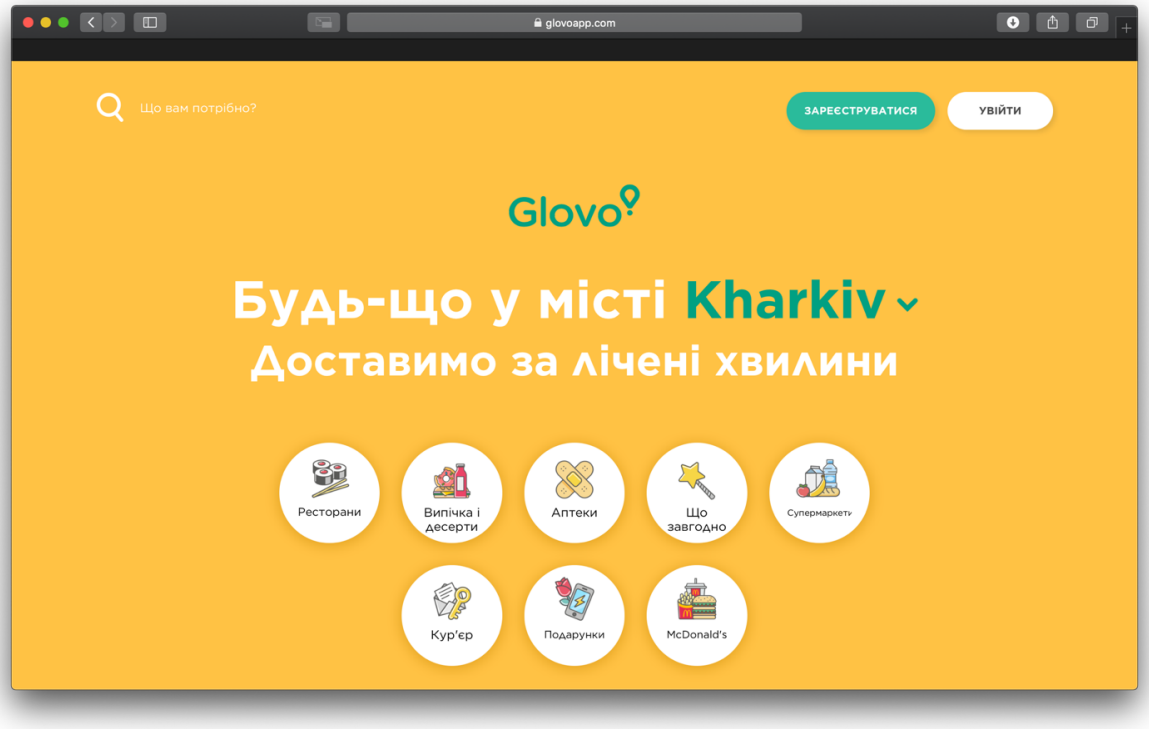

Рисунок 1.1 – Інтерфейс користувача в Glovo

Іншим сервісом для доставки їжі є Uber Eats [16] (див. рис. 1.2).

Це підрозділ компанії Uber - американська компанія, що створила однойменний мобільний застосунок для пошуку, виклику та оплати таксі або приватних водіїв. Застосунок доступний у більш ніж 200 великих містах в 67 країнах світу. На відміну від Uber, він відповідає не за організацію таксі для приватних водіїв, а за доставку їжі. Uber Eats працює в екосистемі компанії, тобто для використання сервісу потрібен обліковий запис Uber.

Та замовлення оформлюють через окремий додаток Uber Eats для мобільних платформ iOS та Android. Uber eats не має сайту, через який можна здійснити замовлення, для цього потрібно мати мобільний додаток, що є великим недоліком.

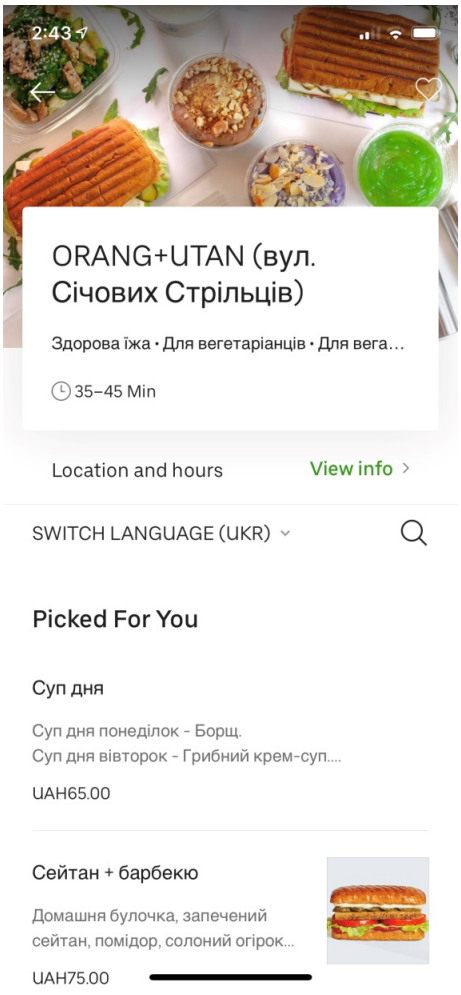

Рисунок 1.2 – Інтерфейс користувача в Uber Eats

Розглянуті сервіси орієнтовані на доставці з їжі з закладів, але не з мереж закладів харчування, тобто в них нема способів оптимізації замовлення, та вирішення проблем із перевантажень замовлень на один заклад, коли є інші вільні. Великі замовлення не поділяються на маленькі замовлення.

1.3 Постановка задачі

Метою цієї роботи є підвищення ефективності бізнес-процесів в системі підтримки мережі онлайн-закладів харчування.

Об’єктом дослідження є задачі лінійного програмування для моделювання бізнес-логіки роботи онлайн-закладів харчування.

Для вирішення практичнох задачі підтримки роботи онлайн-закладів харчування необхідно створити програмну систему на основі клієнт-серверної архітектури, яка буде складатися з наступних частин:

- веб-додатку для клієнтів, через який вони можуть побачити список закладів та замовляти бажані страви. Адміністрування додатку буде проводитись теж, через цей сайт. Після авторизації користувача із правами адміністратора, можна буде додавати інформацію про нові заклади харчування в мережі, про страви з меню, тощо;

- мобільний додаток для водіїв на базі операційної системи Android;

- сервер додатку, на якому буде реалізована бізнес-логіка процесу обслуговування онлайн-замовлень з використанням оптимізаційних ЗЛП;

- сервер БД, на якому буде зберігаться інформація необхідна для поботи програмних додатків та серверу додатку.

Програмний продукт який буде створено, повинен реалізовувати наступні функції:

− реєстрація та авторизація користувачів та водіїв;

− додавання, редагування та видалення необхідної інформації стосовно закладів харчування, їх меню, замовлень, тощо;

− оформлення замовлення за обраними стравами користувачем;

− підтримка основних бізнес-процесів з замовлення та доставки їжі, наприклад, оптимізація замовлення системою шляхом розподілу його для приготування між декількома закладами харчування;

− відправка повідомлень водіям про доставку, прийняття або відхилення пропозиції доставки для водіїв-фрілансерів;

− редагування персональної інформації у мобільному додатку та інше.

Система повинна відповідати основним вимогам, які висуваються до програмних продуктів (стійкість, захист даних при їх введенні, оптимальне використання ресурсів). Інтерфейс користувача має бути ергономічним та

інтуїтивно зрозумілим. Також потрібно використати можливості сучасних смартфонів та хмарних технологій. Смартфони дозволяють відстежувати геолокаційні дані користувача, а отже це може бути використано для відстеження локації водія-кур'єра.

Таким чином, протягом виконання атестаційної роботи магістра необхідно виконати такі задачі:

– провести аналіз бізнес-логіки та моделювання предметної області роботи мережі онлайн-закладів харчування;

– провести досліждення задач лінійного програмування та методів їх вирішення, обрати найкращі для моделювання основних процесів бізнес-логіки мережі онлайн-закладів харчування;

– розробити математичні моделі ЗЛП для виявлених основних процесів бізнес-логіки;

– розробити схему бази даних для збереження інформації предметної області та розроблених математичних моделей;

– розробити алгоритми вирішення промодельованих ЗЛП;

– розробити архітектуру та програмно реалізувати систему підтримки роботи онлайн-закладів харчування.

2 ФОРМУВАННЯ ВИМОГ ДО ПРОГРАМНОЇ СИСТЕМИ

2.1 Призначення розробки

Програмна система призначена для поліпшення сервісу доставки їжі. Головною відмінністю від конкурентів є організація доставки їжі з підтримкою такої логіки, як приготування великих замовлень, ефективнє розподілення замовлень між водіями-кур'єрами та інше. Мобільний додаток буде призначений для користування водіями. Веб-додаток необхіден для замовника їжі, де можна проглянути меню закладу та замовити їжу.

2.2 Вимоги до програмного продукту

Програмна система повинна мати клієнт-серверну архітектуру. Перш за все, потрібно розробити сервер додатку, в якому будуть вирішуватися оптимізаційні задачі для реалізації бізнес-логіки мережі онлайн-закладів харчування. Також сервер має мати можливість комунікувати з геолокаційними сервісами Google: Google Maps, Google Directions та Google Distance Matrix. Обов'язково сервер має бути максимально захищений від несанкціонованого доступу до ресурсів сторонніх осіб. Це необхідно реалізувати за допомогою протоколу авторизації користувачів Oauth2.

Мобільний додаток для водіїв-кур'єрів повинен мати обов'язковий функціонал для визначення поточної геопозиції, вміти працювати з мапами та відображати маршрути. Веб-додаток для замовників їжі повинен мати обов'язковий функціонал для визначення огляду меню закладу, замовлення, вміти стежити за станом виконання замовлення.

Важливим кроком є забезпечення взаємодії серверної складової з клієнтською. Для цього слід використати HTTP REST, який є домінуючим у

середовищі Інтернет. Також варто зазначити, що програмна система має використовувати Firebase сервіси, а саме: Firebase Cloud Messaging та Firebase Realtime Database. Перший для відправки миттєвих повідомлень водіям. Другий використовує NoSQL базу даних, яка дозволяє за лічені мілісекунди сповіщувати своїх підписників про події з даними. Ця служба необхідна для максимально ефективної взаємодії мобільного додатку та серверу, адже вона дозволяє не надсилати HTTP POST запити до серверу, коли наприклад змінюється геолокація автомобіля водія.

Веб-додаток має бути розроблено за допомогою мови програмування Java, за шаблоном MVC з використанням бібліотеки Spring MVC [17]. MVC - це патерн проектування веб-додатків, який включає в себе кілька дрібніших шаблонів. Основна мета застосування MVC полягає в розділенні даних і бізнес-логіки від візуалізації (зовнішнього вигляду). За рахунок такого поділу підвищується можливість повторного використання програмного коду і спрощується супровід.

Веб-сайт буде створено як окремий та незалежний веб-застосунок, через який користувач за допомогою графічного інтерфейсу зможе зробити заказ. Для реалізації веб-сайту, буде використане мови програмування TypeScript, який надалі за допомогою компілятора транслюється у мову JavaScript. Буде використаний фреймворк Angular 7 [18].

Взаємодіяти сервер та веб-застосунок буде за допомогою архітектури REST. Для роботи із базою даних буде використана система управління базами даних PostgreSQL [19]. PostgreSQL - це об'єктно-реляційна система управління базами даних заснована на POSTGRES. PostgreSQL - СУБД з відкритим вихідним кодом, основою якого був код, написаний в Берклі. Вона підтримує більшу частину стандарту SQL і пропонує безліч сучасних функцій.

Сервер буде підключатись та взаємодіяти із базою даних за допомогою Spring Data JPA. Spring Data - додатковий зручний механізм для взаємодії з сутностями бази даних, організації їх в репозиторії, вилучення даних, зміна, в яких випадках для цього буде достатньо оголосити інтерфейс і метод в ньому,

без імплементації. Для відстеження змін у базі даних буде використовуватися Liquibase. Liquibase - це незалежна бібліотека з відкритим вихідним кодом для відстеження, управління та застосування змін схеми бази даних.

2.3 Вимоги до клієнта

Система розробляється як комплекс із серверу додатку, серверу БД, веб-застосунку та мобільного додатку.

Очікується, що веб-додаток буде розгорнутий на Apache Tomcat 8+. На сервері повинен бути встановлений Java 8 версії. Для якісної роботи серверу знадобиться не менше 8 ГБ оперативної пам’яті. Для доступу до веб-сайту клієнту знадобиться швидкісний інтернет, доступ може проводитись як з телефону, так і із комп’ютерів. Для цього клієнти повинні мати в себе браузер. Для роботи усіх компонентів сайту потрібен браузер Google Chrome або Safari. Спеціальних вимог до апаратного забезпечення не потребується.

Другий клієнт програмної системи - це мобільний додаток підтримки водія для операційної системи Android. Мінімальною версію, яка підтримується, є Android Marshmello. Вона була обрана як мінімальна згідно з офіційними статистичними даними, які вказують, що додаток буде працювати на понад 75%. Клієнт повинен мати стабільний зв’язок з мережою Інтернет. Також він має підтримувати можливість відправки геолокаційних даних. На мобільному додатку водія повинні бути встановлені останні версії Google Services.

3 ОПИС ПРИЙНЯТИХ ПРОЕКТНИХ РІШЕНЬ

3.1 Аналіз бізнес-логіки та моделювання предметної області роботи мережі онлайн-закладів харчування

Проведемо аналіз предметної області та бізнес-логіки, що пов’язана з роботою онлайн-закладів харчування.

Заклади харчування, що належать до однієї мережі, мають декілька підрозділів (далі кафе), які знаходяться за різними адресами, мають однакові виробничі потужності (можливість одночасного приготування однакової кількості порцій) та готують страви згідно з меню для реалізації за замовленнями.

Меню - це перелік розташованих у певному порядку різних страв, із зазначенням категорії страви із зазначенням ціни, виходу, способу приготування і переліку компонентів, що входять до їх складу, а також презентацію пропонованих страв та напоїв. Клієнт — замовник, який може обрати страви з меню закладу харчування та оформити замовлення на доставку за вказаною адресою.

Бізнес-логіку роботи закладів харчування можна представити за допомогою діаграми потоків даних, на якій будуть відображено основні бізнес-процеси та структури даних (див. рис. 3.1). Основними бізнес-процесами в роботі мережі онлайн-закладів харчування виступають:

- обробка замовлень, що включає в себе отримання замовлення від клієнта та обробку його оператором; цей процес можна підтримати шляхом організації відповідного інтерфейсу та функціоналу у веб-додатку;

- розподіл замовлення на приготування – це процес закріплення невеликих замовлень для приготування за окремим закладом харчування або розподілу великого замовлення на приготування між декількома закладами харчування; для підтримки цього процесу необхідно забезпечення оптимального розподілу на базі рішення оптимізаційної задачі;

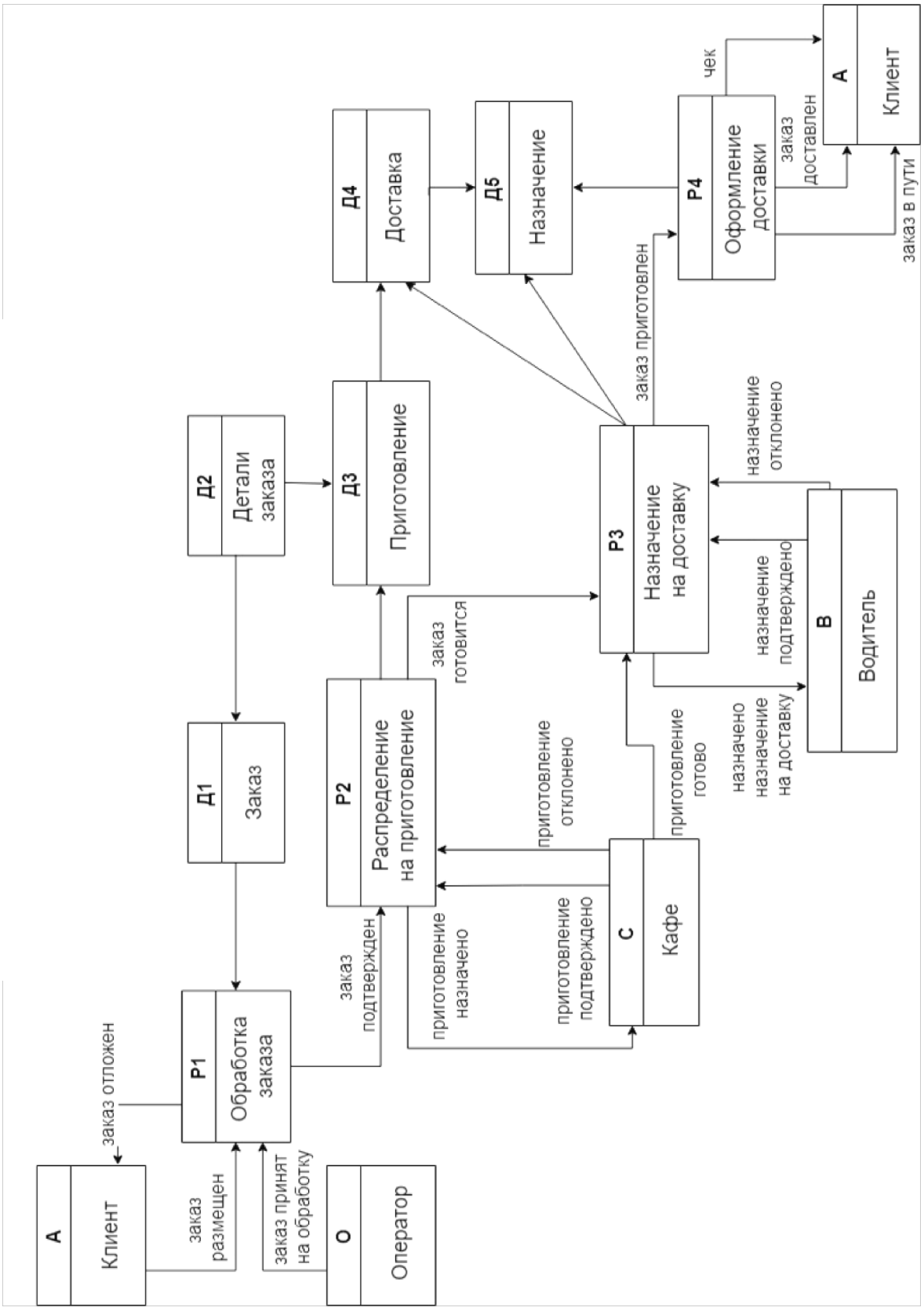

Рисунок 3.1 – Діаграма потові даних

- приготування – процес приготування закладами закріплених частин замовлення; підтримка цього процесу потребую наявності функціоналу для підтвердження закладами харчування закінчення процесу приготування та інше;

- призначення на доставку – це процес, що включає призначення водіїв на доставку замовлення або частин замовлення; для нього необхідна наявність оптимізаційної моделі для оптимального призначення водіїв;

- виконання доставки замовлення водіями; для підтримки цього процесу необхідна наявність відповідного інтерфейсу та функціоналу у мобільного додатку водія.

Отже, існує лише два бізнес-процеси, які потребують алгоритмічної підтримки на базі рішення оптимізаційних задач.

Розглянемо деякі залежності предметної області :

- замовлення можуть виявитися великими, отже їх треба розбивати на окремі роботи з Приготування, які будуть розподілятися між закладами харчування (ЗХ);

- оскільки Замовлення складаються з Деталей (деякої кількості Порцій одного виду страви), то логічно розподіляти на приготування в ЗХ саме ці деталі, по можливості, в як умога меньшу кількість ЗХ;

- невеликі Замовлення (які містять мало Деталей або невелику кількість замовлених Порцій) повинні потрапляти на приготування до одного ЗХ.

Клієнт, який робить замовлення, повинен вказати адресу доставки замовлення та час, на який необхідно виконати замовлення, а при відсутності вказання часу, вважаємо, що замовлення необхідно виконати якомога раніше. Одне замовлення може поєднувати декілька різних порцій страв у різній кількості. Клієнт може неодноразово звертатися до цієї мережі з використанням реєстраційної інформації.

Детальніше розглянемо яким чином зберігається адреса клієнта та доставки. Зазвичай, ці адреса збігаються, але існують випадки, коли постійний клієнт робить замовлення за іншою адресою, тому пропонується зберігати

адресу першого замовлення клієнта як його адресу, а надавати можливість вказання іншої адреси, яка відноситься до конкретного замовлення. Адреса доставки може знаходитися лише у межах певного регіону, у який здійснюється доставка цією мережею. Адреса може бути введена за допомогою вказівки на мапі та подальшою деталізацією (визначенням під'їзду, поверху, квартири / кімнати / офісу), або із введенням назв вулиці, дома подальшою деталізацією. При введені назв вулиць та номерів будинків, будь-яка адреса доставки може бути представлена у вигляді GPS-координат. Таке подання інформації є доцільним для подальшої роботи із доставки замовлень водіями (визначення ЗХ, яке оброблятиме замовлення та маршруту доставки).

Для ЗХ важливою інформацією щодо порцій страв є інгредієнти, та їх кількість, що входять до складу страви, час приготування порції стандартного розміру та кількість порцій страви, які можна приготувати одночасно. За результатами цього аналізу можна побудувати концептуальну модель (КМ) для цієї частини ПО. Основним компонентом концептуальної моделі є опис об'єктів ПО і зв'язків між ними. На рисунку 3.2 показана схема взаємозв'язків об'єктів ПО для бізнес-процесу обробки замовлення.

Рисунок 3.2 - Схема взаємозв'язків основних сутностей з області обробки замовлень

Розглянемо бізнес-процес, що пов'язаний з призначенням водіїв на доставку замовлення:

- приготування з одного ЗХ можна об’єднати в Доставки, бо це частини одного того ж Замовлення;

- потім до кожної Доставки необхідно призначити Водія, який доставить частину замовлення з певного ЗХ за необхідною адресою;

- оскільки Водії можуть мати довільне Розташування (в ЗХ – штатні водії; за межами ЗХ – вільні водії; і, взагалі, будь-який водій може знаходитись десь між ЗХ та адресою замовлення), то отримувати роботу на Доставку Водій буде саме знаходячись в певному Розташуванні.

Отже, в КМ повинні з’явитися нові об’єкти: «Водій», «Доставка», «Приготування» та «Розташування». Схема взаємодії головних об’єктів з призначення водіїв на доставку замовлення представлена на рисунку 3.3.

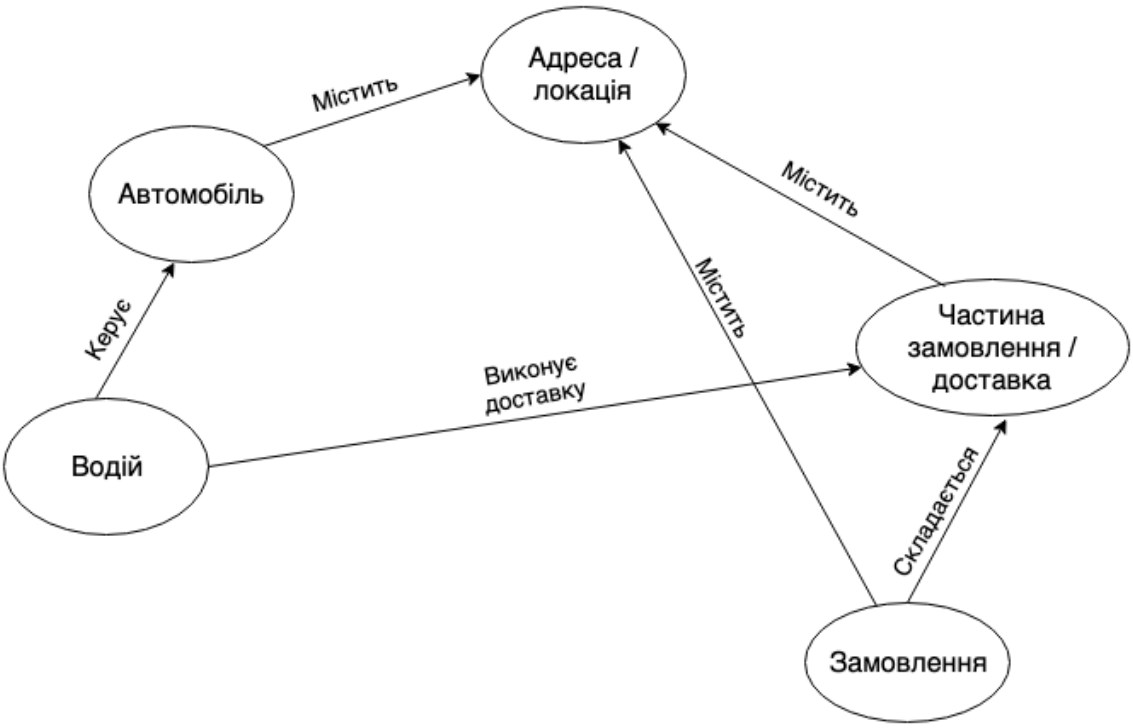

Рисунок 3.3 - Схема взаємодії об’єктів з бізнес-процесу призначення доставки

Основними об’єктами цієї ПО є водії, які виконують доставку замовлень. При реєстрації водія йому надається право доступу до захищених ресурсів програмної системи у вигляді ролі Driver. Таким чином, повинна бути ще й можливість реєстрації водіїв-фрілансерів через мобільний додаток. Оператор,

що обслуговує замовлення, має мати можливість на виконання таких функцій як: додання, редагування, видалення постійних (штатських водіїв).

Водії-фрілансери відрізняються від штатських водіїв тим, що вони мають можливість відмовитися від доставки замовлення. Це перед усім пов’язано з тим, що їх робота з доставки не є основною і вони мають мати можливість обирати для себе найвигідніші маршрути.

Штатні водії навпаки є офіційно працевлаштованими співробітниками. Вони є тою підстраховуючою ланкою, яка буде задіяна, якщо алгоритм не знайде оптимального водія-фрілансера. Штані водії не мають змоги відмовитися від доставки замовлень.

Визначемо атрибути основних сутностей з КМ:

а) «Адреса»: для цього об’єкту зберігаємо інформацію щодо прив’язки розташування певних будівель до GPS-координат, а смае:

1) # Id_адреси – ідентифікатор координат, первинний ключ;

2) GPS-координати – географічні координати точки;

3) країна – країна, адреса якої зберігається;

4) місто – місто, адреса якої зберігається;

5) вулиця – вулиця, адреса якої зберігається;

6) будинок – номер будинка;

7) додаткова_інформація – для зберігання даних, які можуть буди в адресі, наприклад номер квартири, або назва офісної будівлі.

б) «Клієнт»: для цього об’єкту зберігаємо інформацію:

1) #id_клієнта - первинний ключ, ідентифікатор клієнта;

2) імя – ім’я клієнта (за бажанням клієнта він може залишити актуальну інформацію);

3) прізвище – прізвище клієнта (за бажанням клієнта він може залишити актуальну інформацію);

4) логін – ідентифікатор, за допомогою якого клієнт може увійти в систему;

5) пароль – пароль для доступу до інформації щодо власних замовлень клієнта при реєстрації он-лайн;

6) e-mail – електрона пошта клієнта, для зв’язку та інформуванні клієнта;

7) id_адреси – ідентифікатор адреси клієнта;

в) об’єкт «Замовлення» - для нього зберігається інформація щодо замовлення, яке зробив клієнт;

1) #id_замовлення – ідентифікатор замовлення (ключовий атрибут);

2) id_клієнта – ідентифікатор клієнта, який зробив конкретне замовлення;

3) час_замовлення – дата та час, коли було зроблено замовлення;

на який час зам. – дата та час, на який має бути здійснена доставка замовлення, за відсутності вказання часу визначається як найменший час за який може бути зроблена доставка замовлення (включає час приготування страв та час доставки, розраховується за алгоритмами, о писаними далі;

4) бажаний_час_доставки – дата та час, котру клієнт зазначив як бажану при замовленні доставки;

5) # id_адреси – ідентифікатор координат, для фіксації частини адреси, за якою було зроблено замовлення (якщо не вказано іншої адреси – це адреса клієнта).

г) «Страва» – для цього об’єкту зберігаємо інформацію:

1) #id_страви – ідентифікатор страви;

2) Назва – назва страви за меню;

3) час приготування – час, необхідний для приготування порції стандартного розміру з урахуванням наявних напівфабрикатів;

4) час на паралельне приготування – час, необхідний для приготування порції стандартного розміру якщо одна порція вже готується;

5) стандартний розмір на виході – розмір стандартної порції у встановлених одиницях виміру;

6) одиниці вимірювання - одиниці вимірювання розміру порції страви (грами, штуки, мілілітри).

д) об’єкт «Водій» включає всі необхідні дані, а саме: код водія, номер телефону, ім’я, прізвище, електрона пошта, код автомобіля, є фрілансером (так / ні), є вільним (так / ні), зарплатний коефіцієнт;

е) об’єкт «автомобіль» включає: код автомобіля, виробник, модель, номер, координати поточного перебування, максимальна вантажопідйомність (у кілограмах), максимальний об’єм вантажу (у літрах), кількість вільних кг вантажопідйомності, кількість вільних літрів об’єму для вантажу;

ж) об’єкт «частина замовлення» включає в себе наступну інформацію: код частини, дата та час коли буде виконана, адреса, вага, об’єм.

Інформаційна система має наступні обмеження цілісності даних:

- один водій може здійснити багато доставок (1:∞);
- одне замовлення містить декілька частин (1:∞);
- один водій керує одним автомобілем (1:1);
- одна адреса може бути використана у багатьох частинах замовленнь (1:∞).

Для більш повного уявлення про програму і розширеного розуміння завдання, були побудовані наочні UML діаграми [20], які дають більш повний опис програмного продукту, який буде створений в графічному вигляді.

Діаграма прецедентів – діаграма, яка описує функціональне призначення системи або, іншими словами, те, що система буде робити в процесі свого функціонування. На рисунку 3.4 представлена діаграма прецедентів для бізнес-процесу доставки замовлення.

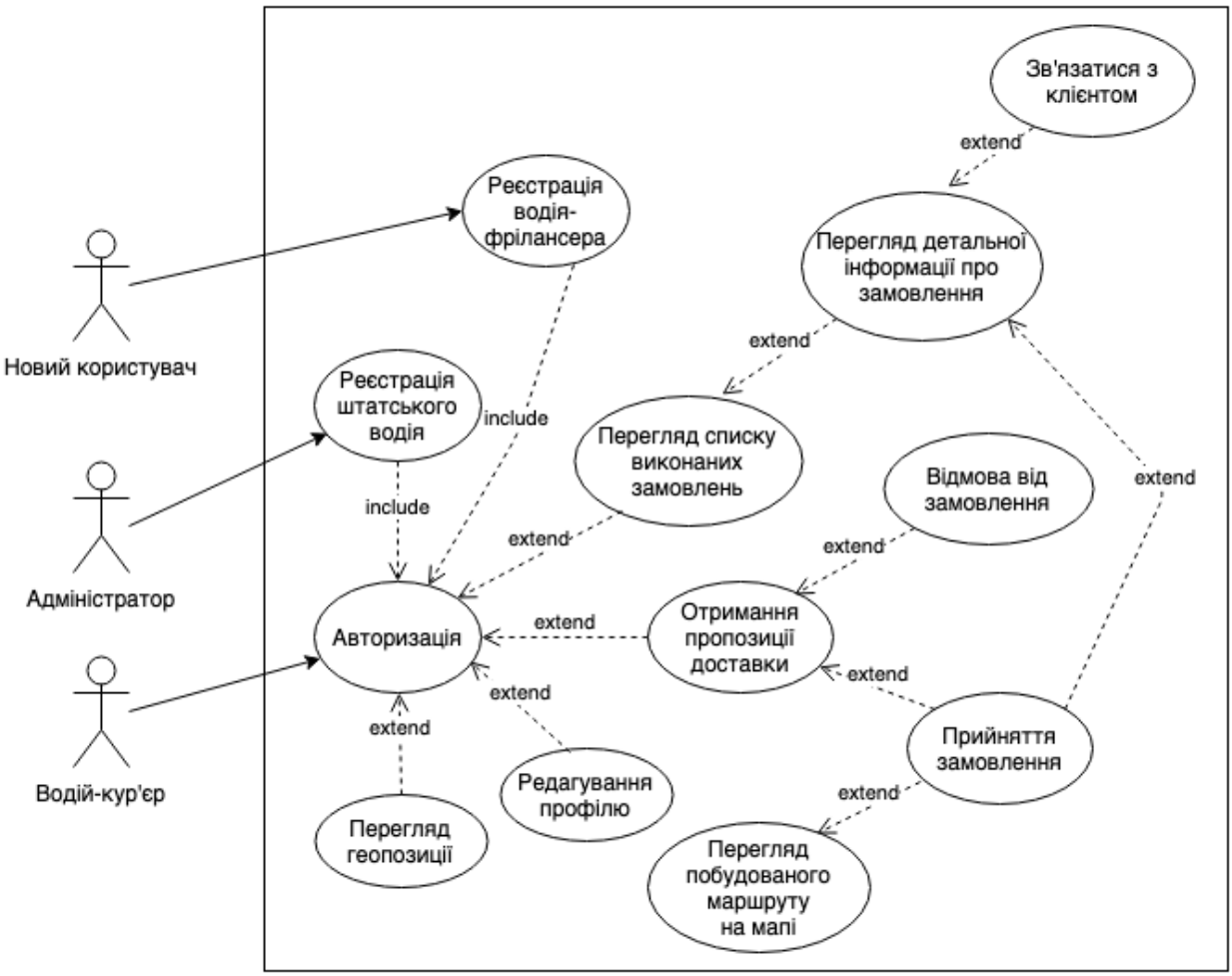

Рисунок 3.4 – Діаграма прецедентів доставки замовлення

Use-case діаграма для обробки замовлення представлена на рисунку 3.5.

Ідентифікація кінцевих подій. Кінцевою подією для клієнта – є створення замовлення, для адміністратора – відредаговане меню. Адміністратор – це свого роду оператор, який контролює ті слідкує за інформацією в системі, передбачається, що адміністратор не тільки переглядає та редагує інформацію, а також слідкує за замовленнями, та їх станом, а якщо треба контактує із закладами мережі харчування.

Розроблені Use-case діаграма дозволять більш адекватно реалізувати підтримку бізнес-процесів в програмній системі.

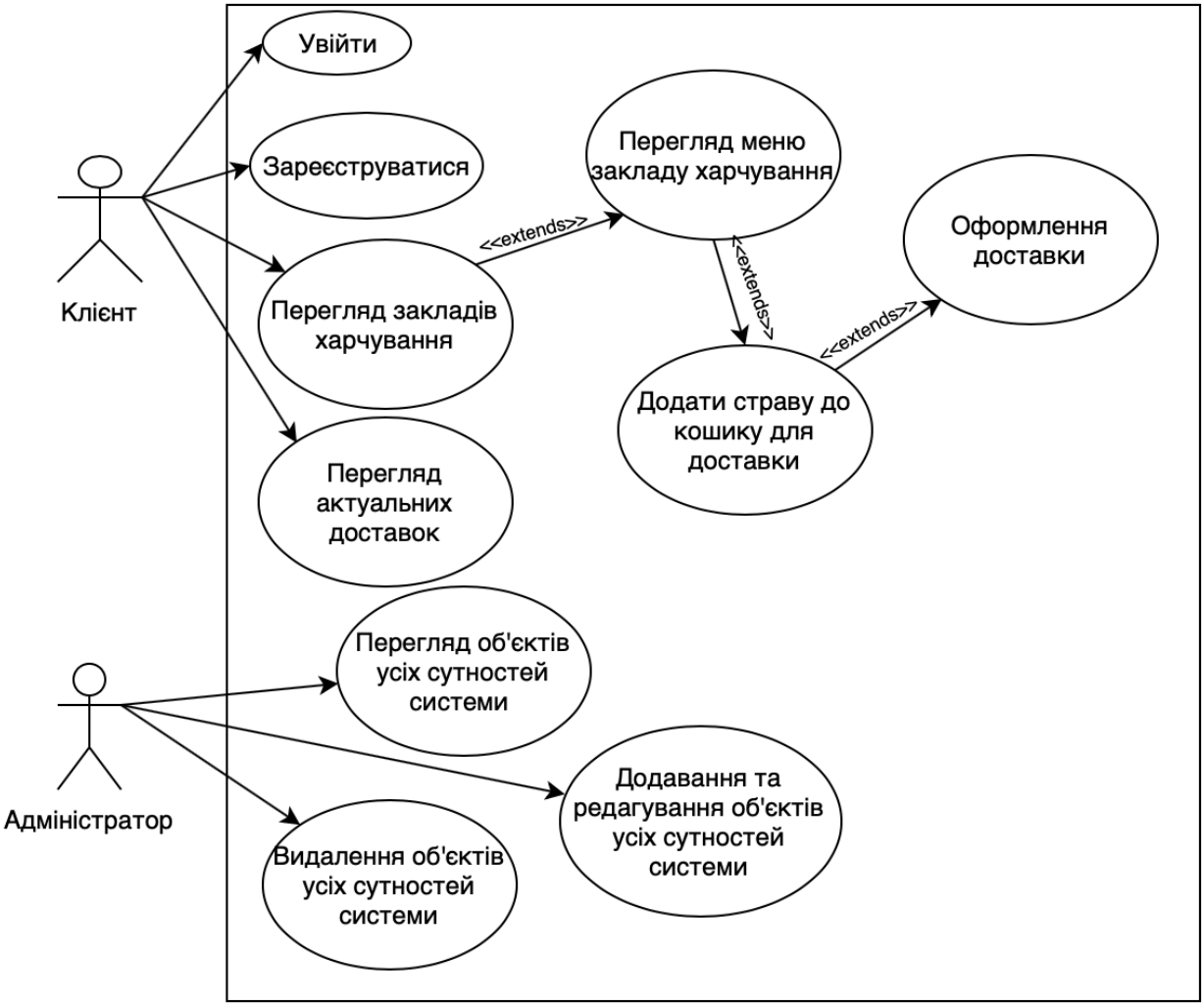

Рисунок 3.5 - Use-case діаграма для обробки замовлення

Виявлені під час аналізу бізнес-процесів та концептуального моделювання сутності та їх атрибути будуть враховані під час проектування бази даних.

3.2 Дослідження задач лінійного програмування

Задачі математичного програмування вирішуються на замовлення керівників: знайти найкраще (оптимальне), єдино вірне і науково обгрунтоване

рішення. Отримавши таке замовлення аналітик досліджує систему, зовнішнє середовище і намагається знайти адекватну модель.

На сьогодні серед ЗМП особливо на практиці виділяються задачі лінійного програмування (ЗЛП) – це задачі, в яких всі функції в моделі лінійні. Всі ЗЛП ще називають оптимізаційними задачами розподілу ресурсів [7, 21].

Оптимізаційні задачі розподілу ресурсів виникають, коли, по-перше, існує ряд заходів, які необхідно виконати, і ряд шляхів їх виконання; по-друге, обмежена кількість ресурсів, недостатня для виконання всіх робіт. Задачу розподілу ресурсів зручно представляти в табличному вигляді (див. табл. 3.1).

Таблиця 3.1 – Надання оптимізаційної задачі розподілу ресурсів

	Роботи						Об'єм наявних ресурсів R_i
Ресурси	J_1	J_2	…	J_j	…	J_n	
R_1	C_{11}	C_{12}	…	C_{1j}	…	C_{1n}	b_1
…	…	…	…	…	…	…	…
R_i	C_{i1}	C_{i2}	…	C_{ij}	…	C_{in}	b_i
…	…	…	…	…	…	…	…
R_m	C_{m1}	C_{m2}	…	C_{mj}	…	c_{mn}	b_m
Об'єм необхідних ресурсів R_i	a_1	a_2	…	a_j	…	a_n	

Загальне формулювання цієї задачі таке: відомі об'єми наявності кожного i-го типу ресурсів bi, об'єми необхідних ресурсів для виконання кожного із j-х типів робіт aj, питомі витрати i-го типу ресурсу на виконання j-го типу робіт Cij. Потрібно знайти такий розподіл ресурсів за роботами, щоб мінімізувалися

сумарні витрати на виконання усіх робіт, або максимізувався одержуваний у підсумку загальний прибуток.

Отже, в оптимізаційній моделі використовують:

- параметри Cij - витрати або прибуток, щодо виділення однієї одиниці ресурсу Ri до роботи Jj;
- шукані невідомі Xij, зазвичай це об'єм і-го ресурсу, необхідного для виконання j-ї роботи.

Загальні витрати (або прибуток) на використання і-го ресурсу для виконання j-ї роботи дорівнюють Xij*Cij, отже маємо лінійну розподільну задачу з цільовою функцією .

$$F = \sum_{j=1}^{n} \sum_{i=1}^{m} C_{ij} * X_{ij} \rightarrow extremum$$

Система обмежень може бути записана:

$$\sum_{j=1}^{n} X_{ij} \leq b_i, \quad \forall i = \overline{1,m}$$

$$\sum_{i=1}^{m} X_{ij} \geq a_j, \quad \forall j = \overline{1,n}$$

$$X_{ij} \geq 0$$

За даною моделлю може бути промодельована задач з розподілу великих замовлень на приготування між різними ЗХ.

В залежності від обмежень в моделі ЗЛП існує відповідна класифікація ЗЛП [23, 24], які можуть використовуватися для вирішення окремих бізнес-задач, наприклад:

- задачі про призначення;
- задачі про суміші;
- -задачі про розкрій або розпіл;
- транспортні здачі та інші.

Якщо $a_j = b_i = 1$ за всіх значень і та j (крім того, що всі $X_{ij} = 1$ або $X_{ij} = 0$), то ми маємо задачу про призначення.

У задачах цього класу для виконання кожної роботи існує потреба одного і лише одного виду ресурсу, а кожний ресурс може бути використаний для однієї і лише однієї роботи. Таким чином, ресурси неподільні між роботами, а роботи неподільні між ресурсами.

Отже, за $a_j = b_i$ та $X_{ij} = 1$ або $X_{ij} = 0$ (ресурси неподільні) узагальнена модель розподілу ресурсів перетворюється до задачі про призначення, а саме:

$$\sum_{j=1}^{n} X_{ij} = 1, \quad \forall i = \overline{1,m}$$

$$\sum_{i=1}^{m} X_{ij} = 1, \quad \forall j = \overline{1,n}$$

$$X_{ij} = 1 \text{ або } X_{ij} = 0$$

За допомогою цієї моделі може бути промодельована задача про призначення водіїв на доставку замовлення.

Отже, для моделювання виділених раніше бізнес-процесів нам підійде або класична задача розподілу ресурсів, або задача про призначення.

3.3 Розробка моделей оптимізаційних задач бізнес-логіки

Однією із виявлених бізнес-логік є розподіл великого замовлення між ЗХ для приготування, які своєчасно повинні виконати замовлення і знизити витрати на доставку частин замовлення від різних ЗХ. Цей процес має за мету якомога знизити витрати на доставку та приготування, за допомогою оптимізації замовлення шляхом розбиття замовлення на декілька маленьких.

Розглянемо залежності, що існують в ПО, які необхідно буде врахувати в оптимізаційній моделі :

- замовлення можуть виявитися великими, і ми будемо розбивати замовлення на окремі роботи з Приготування, які будуть розподілятися між ЗХ;

- оскільки Замовлення складаються з Деталей (деякої кількості Порцій одного виду), то логічно розподіляти на приготування в ЗХ саме ці деталі, по можливості, в як умога меньшу кількість ЗХ;

- невеликі Замовлення (які містять мало Деталей або невелику кількість замовлених Порцій) за оптимізаційною моделлю повинні потрапляти, таким чином, на приготування до одного ЗХ.

Виходячи з цього, ми будемо розподіляти не просто Замовлення, а їх Деталі. Наприклад, в одне замовлення може увійти 80 порцій «Піц з салямі», 100 порцій «Картоплі «Фрі», 60 порцій «Кексів» та 100 порцій «Лимонаду». Тоді ми будемо розподіляти саме ці деталі – 80 порцій піц з салямі для приготування між ЗХ та тому подібне.

Під час розподілу деталей для приготування необхідно врахувати наступні обмеження:

- об’єм $count\_P_j$ деталей P_j поточного замовлення ;

- поточна потужність $\mathrm{p}_j^{\mathrm{i}}$ i-го ЗХ з приготування j-ї деталі поточного замовлення; ми вважаємо, що всі ЗХ обладнані однаково, отже мають потенціально однакові потужності, але оскільки ЗХ вже можуть бути завантажені приготуваннями для інших замовлень, то поточні потужності треба розраховувати;

- залишок $Recid_{\mathrm{n}}^{i}$ n-го продукту в i-му ЗХ, оскільки в кожному ЗХ запаси продуктів обмежені і їх може не вистачити для приготування певної кількості порцій певних страв.

Для формалізації мети розподілу замовлення будемо спиратися на наступні міркування:

- краще розподіляти деталі замовлення для приготування в першу чергу між тими ЗХ, що знаходяться як найближче до клієнта, чиє замовлення оброблюється;

- оптимальним розподілом замовлення будемо вважати таке, для якого було залучено мінімум ЗХ, що розташовані найближче до клієнта, тобто сумарна вартість доставок всіх деталей замовлення повинна бути мінімальною.

Поточна потужність p_j^i i-го ЗХ з приготування j-ї деталі поточного замовлення може бути підрахована як

$$p_j^i = \frac{\left(\text{Order\_on\_time} - \text{Free\_time}_i - \text{Delive\_time}_i\right) * \text{Count\_all\_p}_j}{\text{Cook\_time\_p}_j},$$

де Order_on_time - час, на який було замовлено;

Delive_time_i – час на доставку від і-го ЗХ до адреси, за якою було замовлено;

Free_time_i - час, коли і-тий ЗХ звільниться від приготування аналогічних блюд з інших замовлень;

Cook_time_p_j– час на приготування 1-еї порції страви P_j;

Count_all_p_j – кількість порцій, які можуть бути приготовані одночасно за визначений для страви час.

Розглянемо приклад розподілу деталей між ЗХ.

Для тестового прикладу будемо вважати, що від клієнта отримали замовлення, що складається з наступних деталей:

- піца з салямі (80 порцій);
- картопля «Фрі» (100 порцій);
- кекс (60 порцій);
- лимонад (100 порцій).

В таблиці 3.2 наведено тестовий приклад заповнення табличного надання оптимізаційної задачі з розподілу деталей замовлення між ЗХ та дані з існуючих обмежень.

Опишемо вихідні дані задачі:

а) ЗХ (Cafe - C) $C = \{C_i\}_{i=1}^{\text{count\_C}}$ –- це наші ресурси, за допомогою яких ми зможемо приготувати деталі замовлення;

б) деталі D_j^k замовлення Z^k (Details – D, де $Z^k = \{D_j^k\}_{j=1}^{\text{count\_dk}}$) - це роботи, що необхідно розподілити між ЗХ для приготування; кожна деталь замовлення складається з count_P_j кількості порцій P_j одного виду: $D_j^k = \langle P_j, \text{count\_P}_j \rangle$;

в) вартість $Price_{ik}$ – вартість доставки замовлення з *i*-го ЗХ за адресою, на яку було зроблено *k*-те замовлення.

г) кількість $count\_P_j$ порцій одного виду P_j деталі поточного замовлення , яку необхідно приготувати;

д) поточні потужності p_j^i *i*-их ЗХ з приготування *j*-их деталей замовлення.

Таблиця 3.2 - Тестові дані до оптимізаційної моделі (обмеження по потужності ЗХ)

	Деталі замовлення,які необхідно приготувати			
Заклади харчування	Піца з салямі	Картопля «Фрі»	Кекс	Лимонад
ЗХ № 1	Max 10 порцій	Max 10 порцій	Max 40 порцій	Max 60 порцій
ЗХ № 2	Max 50 порцій	Max 50 порцій	Max 40 порцій	Max 70 порцій
ЗХ № 3	Max 50 порцій	Max 50 порцій	Max 40 порцій	Max 60 порцій
ЗХ № 4	Max 50 порцій	Max 50 порцій	Max 20 порцій	Max 70 порцій
ЗХ № 5	Max 50 порцій	Max 30 порцій	Max 20 порцій	Max 70 порцій
Об’єм необхідних порцій	80 порцій	100 порцій	60 порцій	100 порцій

Розрахунок вартості $Price_{ik}$ доставки з *i*-го ЗХ за адресою, на яку було зроблено *k*-те замовлення, будемо проводити за формулу:

$$Price_{ik} = Distance_{ik} * avg_{l=1}^{count\_Dr}(Price_l),$$

де $Distance_{ik}$ – відстань у км від місця знаходження і-го ЗХ ($Location\_C_i$) до місця доставки *k*-того замовлення ($Location\_Z_k$);

$avg_{l=1}^{count\_Dr}(Price_l)$ – середня вартість кілометру доставки серед водіїв.

Для побудови оптимізаційної моделі:

а) визначимо керовані змінні, які будемо шукати під час вирішення оптимізаційної задачі: X_{ij} – змінна, що показує яку кількість порцій деталі D_j^k повинен приготувати ЗХ C_i ;

б) визначимо обмеження:

1) $\sum_{i=1}^{count\_C} X_{ij} \geq count\_P_j, \ \forall j = \overline{1, count\_D_k}$ - означає, що кожна деталь замовлення D_j^k повинна бути приготовлена в кількості не менше ніж було замовлено ($count\_P_j$);

2) $0 \leq X_{ij} \leq p_j^i, \ \forall i = \overline{1, count\_C}, \ \forall j = \overline{1, count\_D_k}$ - означає, що кількість X_{ij}, що буде розподілена для приготування в ЗХ повинна бути позитивною та не повинна перевищувати поточну потужність p_j^i ЗХ з приготування цього типу порцій;

в) визначимо мету оптимізаційної задачі: нам необхідно досягти найменших витрат та часу на доставку цього великого замовлення до клієнта, отже чим більше порцій буде розподілено для приготування до найближчого ЗХ, тим краще.

Описана мета оптимізаційної задачі доставки може бути сформульована наступним чином:

$$F = \sum_{i=1}^{count\_C} (Price_{ik} * Sgn\ (\sum_{j=1}^{count\_D_k} X_{ij}\ )) \rightarrow min$$

де $Price_{ik}$ вартість доставки з i-го ЗХ за адресою, на яку було зроблено k-те замовлення.

$Sgn\left(\sum_{j=1}^{count_{D_k}} X_{ij}\right)$ - буде приймати значення 1, якщо буде існувати хоча б одне $X_{ij} \neq 0$ в i-ій строчці (тобто існує необхідність доставки з C_i ЗХ).

Отже, заключна версія оптимізаційної моделі може бути сформульована наступним чином:

$$F = \sum_{i=1}^{count\_C} (Price_{ik} * Sgn\,(\sum_{j=1}^{count\_D_k} X_{ij})) \rightarrow min$$

$$\sum_{i=1}^{count\_C} X_{ij} \geq count\_P_j, \quad \forall\, j = \overline{1, count\_D_k}$$

$$0 \leq X_{ij} \leq p_j^i, \qquad \forall\, i = \overline{1, count\_C}, \qquad \forall\, j = \overline{1, count\_D_k}$$

В таблиці 3.3 наведено результати вирішення оптимізаційної задачі з розподілу деталей замовлення між ЗХ з урахуванням існуючих обмежень.

Задача розподілу водіїв між роботами з доставки замовлень клієнтам відноситься до класичної задачі про призначення.

Опишемо вихідні дані моделі:

– множина Водіїв (Driver - Dr) $\mathrm{Dr} = \{\mathrm{Dr_l}\}_{\mathrm{l=1}}^{\mathrm{count\_Dr}}$, що доставляють частини або повні Замовлення;

– множина робіт (Job - J) з доставки $J = \{J_h(i,k)\}_{\mathrm{h=1}}^{\mathrm{count\_J}}$; кожна робота $J_h(i,k)$ характеризується двома параметрами: i – номер ЗХ, де була приготована і знаходиться доставка; k – номер замовлення, за адресою якого необхідно відвести доставку;

– $Price_{lh}$ – вартість доставки l-тим водієм роботи $J_h(i,k)$;

– обмеження на кількість водіїв, що повинні доставити одну доставку, - одну Доставку доставляє один Водій;

– обмеження на наявних водіїв – Водій може бути призначений на одну Доставку або на жодну.

Таблиця 3.3 - Приклад результатів вирішення оптимізаційної задачі

	Деталі замовлення,які необхідно приготувати			
Потужності кафе	Піца з салямі	Картопля «Фрі»	Кекс	Лимонад
ЗХ № 1	0	0	0	0
ЗХ № 2	50	50	40	70
ЗХ № 3	30	50	20	30
ЗХ № 4	0	0	0	0
ЗХ № 5	0	0	0	0
Об’єм необхідних порцій деталей	80 порцій	100 порцій	60 порцій	100 порцій

Вартість доставки можна підрахувати наступним чином:

$$Price_{lh} = Distance_{lh} * Price_l,$$

де $Distance_{lh}$ – відстань у км від місця знаходження l-го водія ($Location\_Dr_l$) до місця доставки роботи $J_h(i,k)$ $Location\_Z^k$.

$$Distance_{lh} = Map\ (Location\_C_i, Location\_Z^k),$$

де Map – це функція, яка програмно реалізована для роботи з картами та дозволяє знайти відстань у км між двома координатами;

$Location\_C_i$ – координати ЗХ, де знаходится приготована частина замовлення для доставки;

$Location\_Z^k$ – координати замовлення зберігаються в полі «координати» таблиці «Замовлення».

Для побудови оптимізаційної моделі:

– визначимо керовані змінні, які будемо шукати під час вирішення оптимізаційної задачі: X_{lh} – змінна, що показує чи призначено водія $\mathrm{Dr_l}$ на роботу $\mathrm{J}_h(i,k)$ з доставки з i-го ЗХ за k-ю адресою замовлення; отже $X_{lh} = 1$, якщо $\mathrm{Dr_l}$-й водій призначений на роботу $\mathrm{J}_h(i,k)$ з доставки, $X_{lh} = 0$ - у протилежному випадку;

– визначимо мету оптимізаційної задачі: візьмемо за мету доставити всі роботи з мінімальними витратами на доставку. Ми знаємо $Price_{lh}$ – вартість доставки l-тим водієм роботи $J_h(i,k)$ до її клієнта. Отже мета оптимізаційної задачі може бути виражена наступним чином:

$$F = \sum_{l=1}^{count_{Dr}} \sum_{h=1}^{count_J} Price_{lh} * X_{lh} \rightarrow \min\ ;$$

– визначимо обмеження, що притаманні задачам про призначення:

1) $\sum_{h=1}^{count\_J} X_{lh} \leq 1,\ \forall\, l = \overline{1, count\_Dr}$ - означає, що кожен водій може виконати нуль або одну доставку;

2) $\sum_{l=1}^{count\_Dr} X_{lh} = 1,\ \forall h = \overline{1, count\_J}$ - означає, що для доставки кожної роботи може бути призначено тільки одного водія;

3) $X_{lh} = 1$ або $X_{lh} = 0$ - означає, що водій може бути або призначений або ні.

В таблиці 3.4 наведено табличне надання задачі про призначення водіїв на роботи з доставки.

Таблиця 3.4 – Табличне надання задачі про призначення Водіїв на роботи з доставки

	Роботи з доставки					Кіл-сть наявних водіїв Dr
Водії	$J_1(1,1)$	…	$J_h(i,k)$	…	$J_{count_J}(.,.)$	
Dr_1	$Price_{11}$	…	$Price_{1h}$	…	$Price_{1count\_J}$	1
Dr_2	$Price_{21}$		$Price_{2h}$	…	$Price_{2count\_J}$	1
…	…	…	…	…	…	…
Dr_l	$Price_{l1}$	…	$Price_{lh}$	…	$Price_{lcount\_J}$	1
…	…	… .	…	…	…	…
Dr_{count_Dr}	$Price_{count\_Dr1}$	…	$Price_{count\_Drh}$	…	$Price_{Count\_Drc}$	1
Кіл-сть не обхідних водіїв Dr	1	1	1	…	1	

Введемо додаткові обмеження до оптимізаційної моделі:

– часові обмеження - стосуються того, що час $Tprob_{lh}$, коли імовірно l-тий водій виконає роботу $J_h(i,k)$ з доставки, не повинен перевищувати час $Torder_h$, на який було замовлення;

– вагові обмеження – стосуються того, чи зможе той чи інший водій вмістити всі порції блюд, що зібрані для доставки.

Імовірний час доставки l-тим водієм роботи $J_h(i,k)$ який ми повинні підрахувати та повідомити клієнтові, $Tprob_{lh}$ повинен враховувати :

– час $Tready_{lh}$, на який ЗХ l зможе виконати роботу з приготування необхідної кількості порцій;

– тривалість $Duration_{lh}$ доставки l-тим водієм роботи $J_h(i,k)$ до її клієнта: її можна вирахувати за домогою сервісу Google Distance Matrix, який

обчислює тривалість подорожі між точками з урахуванням поточної завантаженності доріг.

Отже , ймовірний час доставки *l*-тим водієм роботи $J_h(i,k)$ рахується як:

$$Tprob_{lh} = Tready_{lh} + Duration_{lh} \quad .$$

Тоді, часові обмеження до оптимізаційної моделі будуть виражені як:

$$\sum_{l=1}^{count\_Dr} Tprob_{lh} * X_{lh} \leq \mathrm{T}order_h \quad , \forall h = \overline{1, count\_J}$$

Вагові та об'ємні обмеження стосуються того, чи зможе той чи інший водій вмістити частину замовлення, що необхідно доставити. Отже, у кожного водія є своя вантажопідйомність Wd_l та максимальний об'єм Vd_l. Вантажопідйомність доставки W_h та об'єм V_h отримуються ззовні, тобто вони задаються при запиті на доставку і розраховується підсистемою розподілу навантаження на ЗХ.

Робота $J_h(i,k)$ водія полягає в доставці певних доставок $\mathrm{D_j^k}$ замовлення $\mathrm{Z^k}$ до клієнта. Ці доставки будуть сформовані на основі приготованих порцій у ЗХ та їх кількості.

В свою чергу, m-та доставка k-го замовлення $\mathrm{Del_m^k} = \{\langle \mathrm{D_j^k},\ \mathrm{count\_D_j^k} \rangle_\mathrm{h}\}_{\mathrm{h=1}}^{\mathrm{count\_del_m}}$ складається з $\mathrm{count\_del_m}$ деталей $\mathrm{D_j^k}$ замовлення в кількості $\mathrm{count\_D_j^k}$ порцій.

Таким чином, вантажопідйомність доставки W_h , яку необхідно відвезти водієві, може бути розрахована наступним чином

$$W_h = \sum_{p=1}^{count\_port} \mathrm{Size\_P_j} \ * \mathrm{count\_D_j^k},$$

де $\mathrm{Size\_P_j}$ - розмір j-ої порції деталі $\mathrm{D_j^k}$ замовлення (поле «Розмір» таблиці «Порція»);

$\mathrm{count\_D_j^k}$ – кількість порцій, що увійшли до h-го приготування (поле «Кількість» таблиці «Приготування»).

Відобразимо дані стосовно вагових обмеження в таблиці 3.5.

Таблиця 3.5 - Табличне надання вагових обмежень

	Роботи з доставки					Вагові обме-ження водіїв Dr
Водії	$J_1(1,1)$	…	$J_h(i,k)$	…	$J_{count_J}(.,.)$	
Dr_1	X_{11}	…	X_{1h}	…	X_{1count_J}	Wd_1
Dr_2	X_{21}	…	X_{2h}	…	X_{2count_J}	Wd_2
…	…	…	…	…	…	…
Dr_l	X_{l1}	…	X_{lh}	…	X_{lcount_J}	Wd_l
…	…	…	…	…	…	…
Dr_{count_Dr}	$X_{count\_Dr\ 1}$	…	$X_{count\_Dr\ h}$	…	$X_{count\_Dr\ count}$	Wd_{count_Dr}
Вага доставок	W_1		W_h		W_{count_J}	

Тоді, додаткові обмеження до оптимізаційної моделі будуть:

$$\sum_{h=1}^{count\_J} W_h * X_{lh} \leq Wd_l \quad ,\forall l = \overline{1, count\_Dr}_i$$

Під доставкою слід розуміти частину замовлення. Отже, повне замовлення можна розглядати як множину доставок, що представлено як:

$$\mathrm{Z^k} = \{\mathrm{Del}_m^k(i)\}_{m=1}^{count\_Del_k},$$

де $\mathrm{Del_m^k}(i)$ – це m-та Доставка з i-го ЗХ за адресою k-го Замовлення.

Отже, остаточний варіант моделі представлено за допомогою наступної системи рівнянь:

$$F = \sum_{l=1}^{count\_Dr} \sum_{h=1}^{count\_J} Price_{lh} * X_{lh} \rightarrow min,$$

$$\sum_{h=1}^{count\_J} X_{lh} \leq 1, \qquad \forall\, l = \overline{1, count\_Dr},$$

$$\sum_{l=1}^{count\_Dr} X_{lh} = 1, \qquad \forall h = \overline{1, count\_J},$$

$$\sum_{l=1}^{count\_Dr} \mathrm{T}prob_{lh} * X_{lh} \leq \mathrm{T}order_{h}, \qquad \forall h = \overline{1, count\_J}$$

$$\sum_{h=1}^{count\_J} W_h * X_{lh} \leq Wd_l, \qquad \forall l = \overline{1, count\_Dr}$$

$$\sum_{h=1}^{count\_J} V_h * X_{lh} \leq Vd_l, \qquad \forall l = \overline{1, count\_Dr}$$

$$X_{lh} = 1 \text{ або } X_{lh} = 0$$

Дана математична модель показує цільову функцію та її обмеження, які випливають із реальних (життєвих) складнощів транспортування вантажів.

Введені в побудованих математичних моделях позначення необхідно врахувати під час проектування бази даних.

3.4 Проектування бази даних

Для зберігання даних був обраний реляційний підхід Реляційні бази даних активно використовуються декілька десятків років. Для правильної та безпечної роботи програмної системи були створені відношення, які мають на меті розмежування прав доступу користувачів до захищених ресурсів у системі.

Також були створені таблиці для зберігання даних про об’єкти, що було промодельовано в КМ та в оптимізаційних моделях, побудованих вище.

На рисунку 3.6 представлено фрагмент ER-діаграми, яка відповідає бізнес-задачі обробки замовлень.

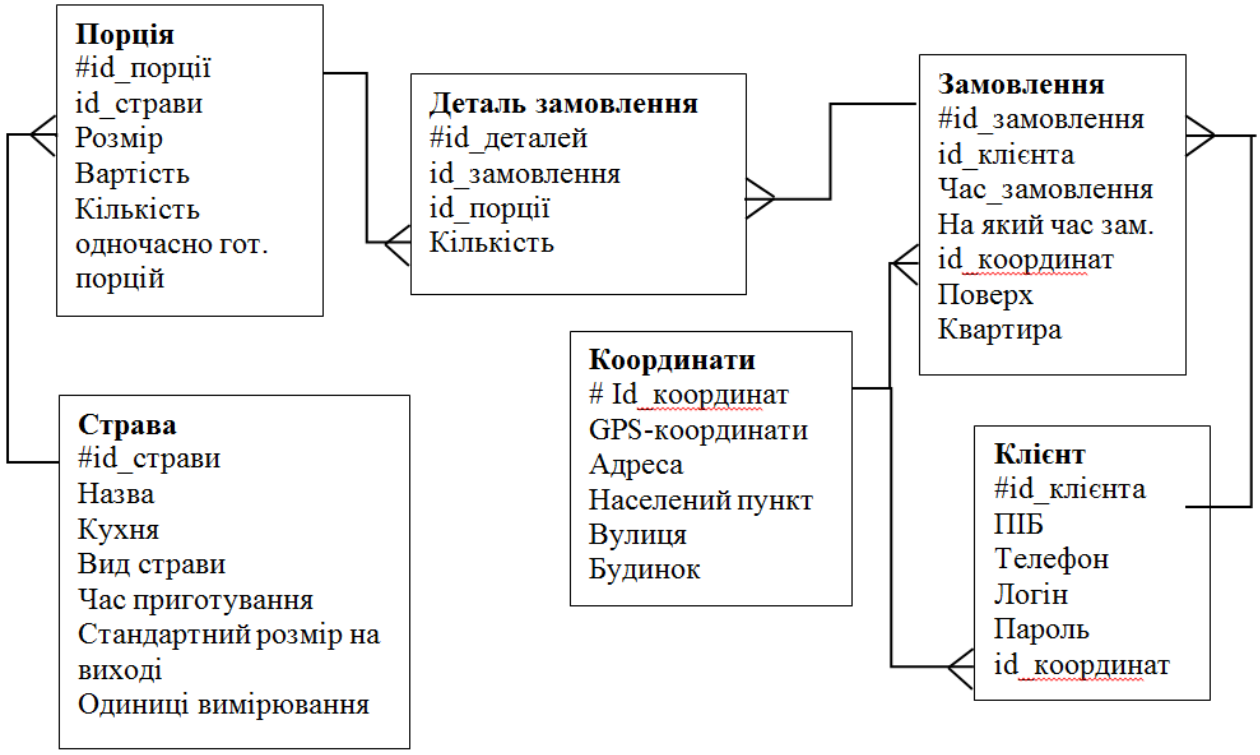

Рисунок 3.6 – ER-діаграма для бізнес-задачі обробки замовлень

На ER-діаграмі показані основні сутності предметної області, їх атрибути та зв’язки. Решіткою позначені первинні ключи сутностей. На базі розробленої діаграми може бупи побудована реляційна схема БД. На рисунку 3.7 наведено фрагмент схеми БД, що відповідає задачі про обробку замовлень.

Всі отримані відношення також знаходиться в 3НФ, оскільки не існує транзитивних залежностей між неключовими атрибутами. Отже, представлена схема БД відповідає вимогам третьої нормальної форми, та на її основі може бути побудована фізична модель бази даних.

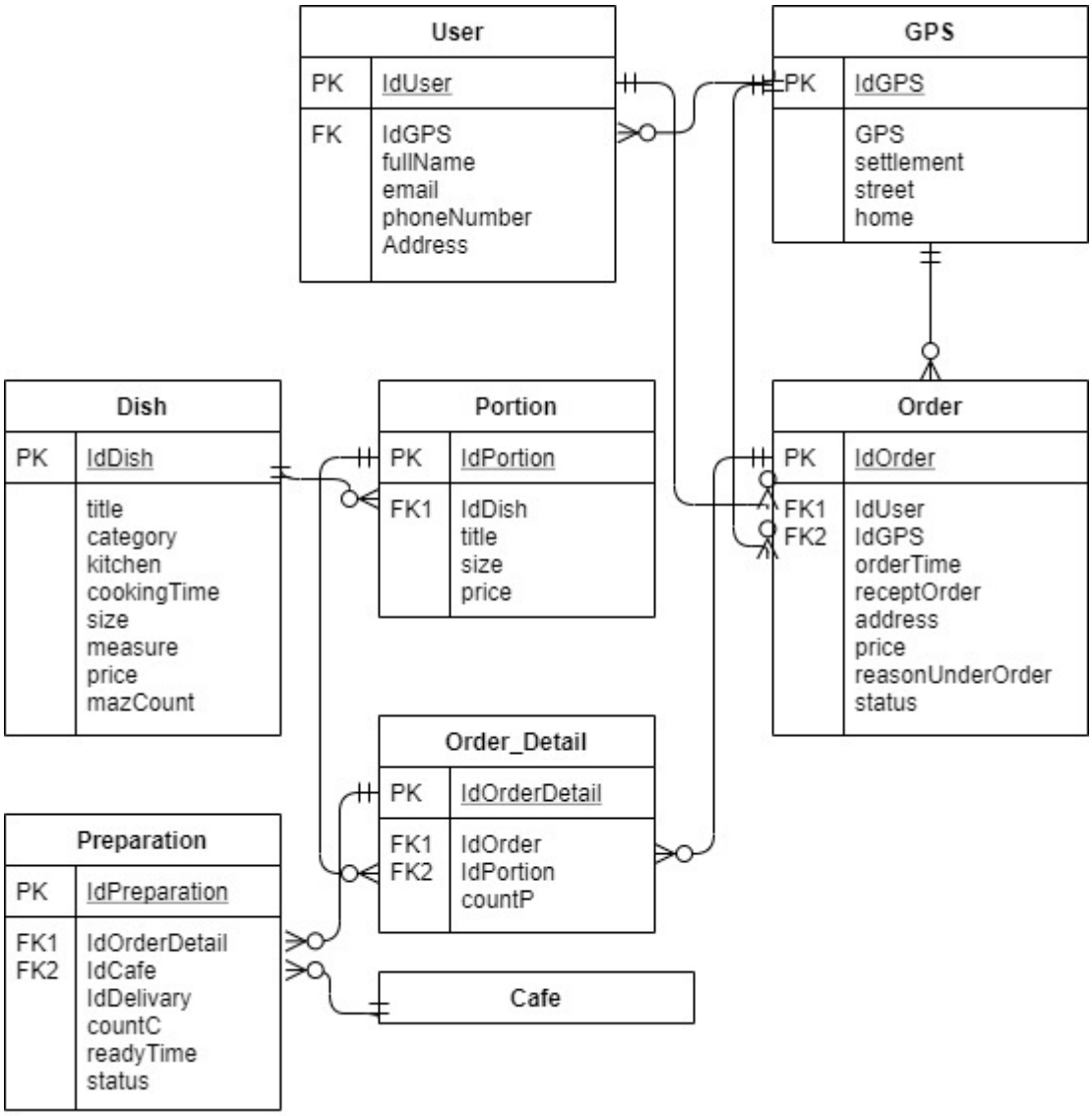

Рисунок 3.7 – Схема БД для бізнес-задачі обробки замовлення

На рисунку 3.8 представлено фрагмент ER-діаграми, яка відповідає бізнес-задачі призначення водіїв на доставку. До діаграми потрапили не тільки об’єкти, що було виділено під час КМ, але й ті дані, що необхідні для збереження розробленої оптимізаційної задачі.

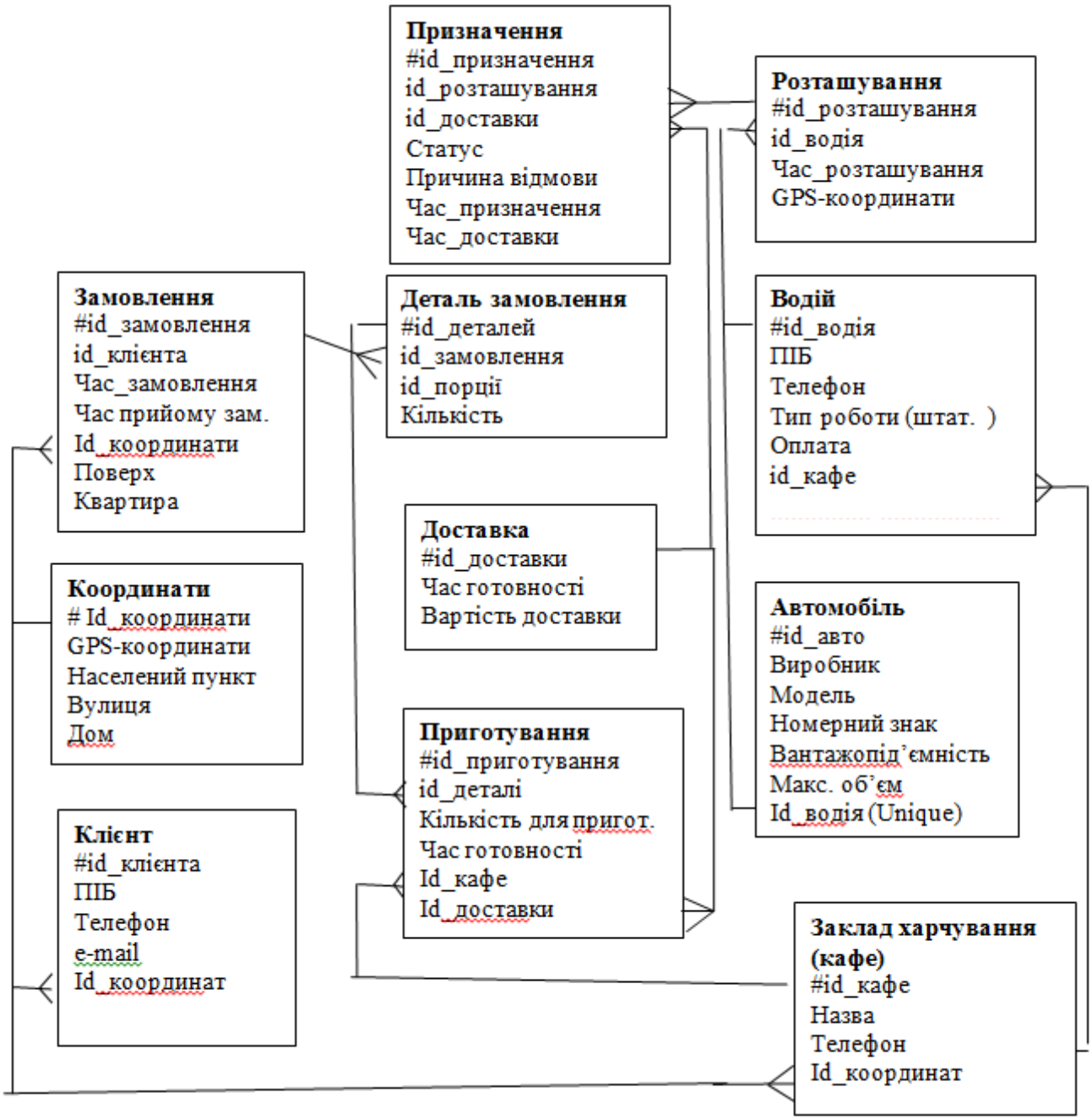

Рисунок 3.8 – ER-діаграма для бізнес-задачі призначення водіїв

Всі відношення в ER-діаграмі знаходиться в 3НФ та на її основі може бути побудована логічна модель реляційної БД та фізична модель бази даних.

3.5 Розробка алгоритму рішення оптимізаційних задач

Для вирішення переважної більшості оптимізаційних задач використовуються різні методи математичного програмування, що дозволяють знайти екстремальне значення цільової функції або приблизне значення цільової функції. Для вирішення лінійної задачі програмування існує декілька методів: графічний метод, симплекс-метод, «жадібний» алгоритм та інші.

Графічне рішення задачі лінійного програмування. Цей метод є досить простим, наочним і дозволяє зробити деякі загальні висновки за рішенням оптимізаційних задач методами лінійного програмування. Однак застосування графічного методу обмежена завданнями відносно невеликий розмірності.

Симплекс-метод є універсальним аналітичним методом вирішення завдань лінійного програмування. Симплекс - поняття геометричне, що означає сукупність вершин багатовимірного тіла. Ідея симплекс-метода полягає в послідовному переборі рішень - в послідовному переході від однієї вершини до іншої. Однак цей перебір не хаотичний, а такий, що на кожному кроці рішення поліпшується. Метод складається з двох етапів: на першому етапі шукається допустиме рішення; на другому етапі це допустиме рішення поліпшується до оптимального.

«Жадібний» алгоритм — метод оптимізації задач лінійного програмування, заснований на тому, що процес ухвалення рішення можна розбити на елементарні кроки, на кожному з яких ухвалюється окреме рішення. Рішення приймається на кожному кроці повинне бути оптимальним тільки на поточному кроці і повинне прийматися без урахування попередніх або подальших рішень.

Для рішення оптимізаційної задачі розподілу деталей замовлення між ЗХ було розроблено алгоритм на базі жадібного алгоритму.

Наведемо опис алгоритму з використанням введених позначень:

- 1-й крок: формування множини ЗХ $C = \{C_i\}_{i=1}^{count\_C}$, за допомогою яких ми зможемо приготувати деталі замовлення; в якості принципу жадібності для даного алгоритму обрано той принцип, що ЗХ потрапляє до множини з урахуванням мінімізації їх відстані до адреси, за якою було зроблено замовлення Z^k ($Distance_{ik}$ – відстань у км від місця знаходження і-го ЗХ ($Location\_C_i$) до місця доставки k-того замовлення ($Location\_Z_k$)) ;

- 2-й крок: формування множини деталей D_j^k замовлення Z^k ($Z^k = \{D_j^k\}_{j=1}^{count\_dk}$), які потім необхідно буде розподілити між ЗХ для приготування; кожна деталь замовлення складається з $count\_P_j$ кількості порцій P_j одного виду: $D_j^k = \langle P_j, count\_P_j \rangle$;

- 3-й крок: обирається найближчий ЗХ з множини $C = \{C_i\}_{i=1}^{count_C}$для призначення в нього деталей на приготування;

- 4-й крок: обирається деталь замовлення для призначення в ЗХ на приготування;

- 5-й крок: ЗХ перевіряється на відповідність обмеженням оптимізаційної моделі. ЗХ, для якого виконуються всі обмеження повною мірою загружається даною деталлю замовлення для приготування з максимальним урахуванням його потужності з цього блюда;

- 6-й крок: якщо деталі замовлення ще не закінчилися, переходимо на крок 4, інакше далі;

- 7-й крок: якщо ЗХ загружений повністю, а деталі замовлення це залишилися, переходимо на крок 3, інакше далі;

- 8-й крок: результатами роботи алгоритму заносимо до бази даних.

На рисунку 3.9 наведено загальну схему роботи алгоритму.

Для вирішення оптимізаційної задачі призначення водіїв на доставку був обраний «жадібний» алгоритм, який здатен вирішити оптимізаційну задачу у найкоротші строки, що є досить важливим для систем реального часу.

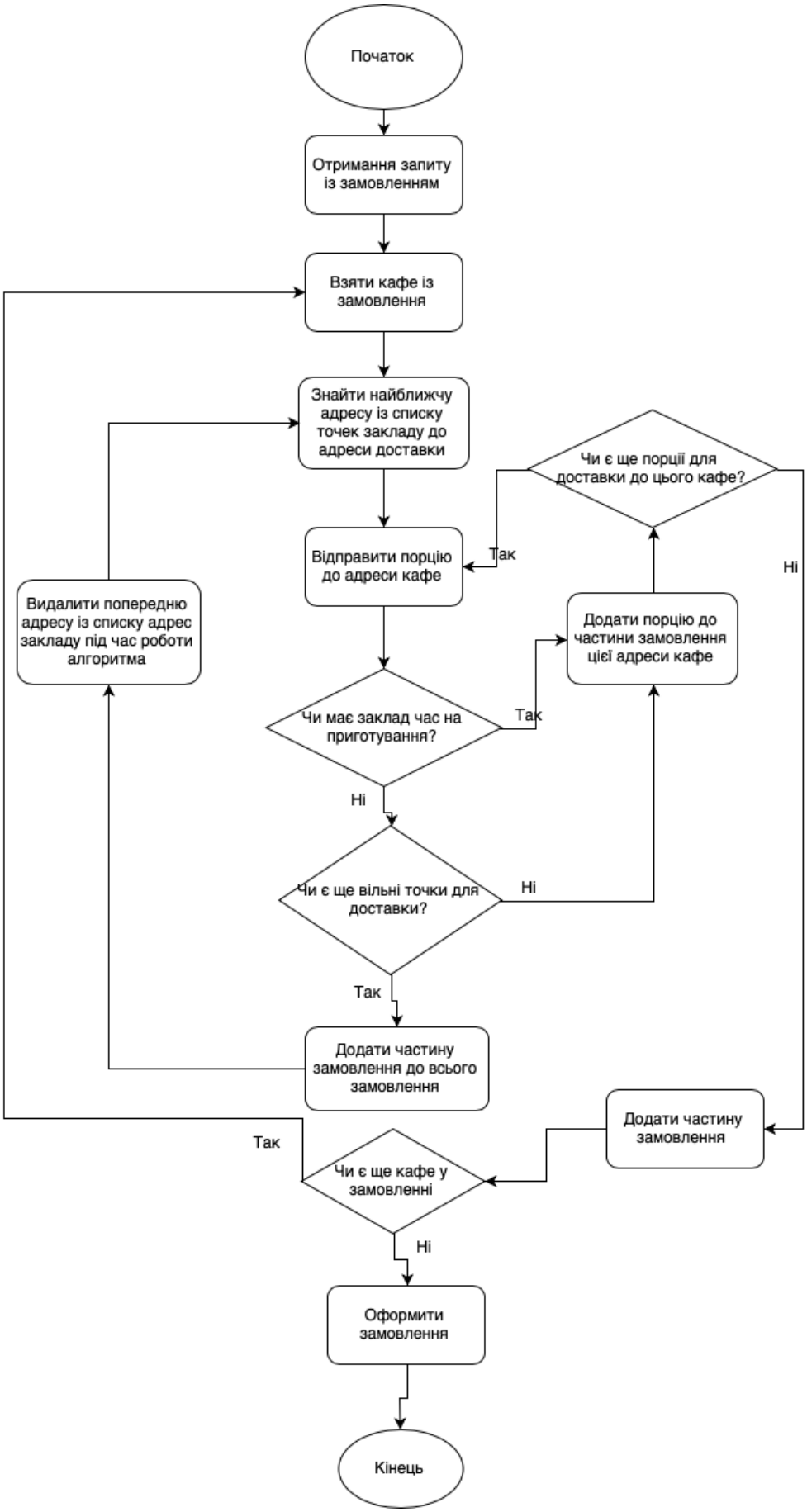

Рисунок 3.9 – Блок схема алгоритму розподілу замовлення

Для простого уявлення про те, як працює алгоритм слід звернутися до рисунку 3.10, на якому зображені основні етапи виконнаня алгоритму.

Рисунок 3.10 – Скорочена блок схема алгоритму

В цьому алгоритмі доцільно враховувати у реальному часі відповідь водіїв-фрілансерів. Такий алгоритм має працювати на основі рекурсії, що дозволяє «жадібно» обирати найоптимальнішого водія на кожному її кроці.

Виходячи з логіки роботи алгоритму можна зробити висновок, що при великих замовленнях алгоритм успішно знаходить найоптимальніших водіїв для різних частин замовлення. Якщо ж замовлення складається з невеликих частин, то для мінімізації вартості доставки алгоритм закріплює одного водія за всіма доставками.

3.6 Розробка архітектури системи

Як вже було зазначено система підтримки мережі онлайн-кафе складається із двох клієнтських додатків: веб-додатку обробки замовлень та мобільного додатку підтримки доставки замовлення водієм (див. рис. 3.11).

У якості архітектури для програмної системи було обрано архітектуру клієнт-сервер в стилі REST. Такий підхід дозволяє швидко вносити зміни до логіки головних бізнес-процесів, які повинні бути зосереджені на серверній складовій. У клієнт-серверній архітектурі виконуються певні правила, які наведені нижче.

Сервери є незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає жорсткої прив'язки клієнтів до серверів. Більш ніж типовою є ситуація, коли один сервер одночасно обробляє запити від різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але можуть не мати жодного уявлення про існування інших клієнтів.

Клієнти ініціюють запити до серверів; сервери обробляють запити і повертають відповідні відповіді. Запити та відповіді створюються на базі передачі уявлень ресурсів. REST спочатку описаний в контексті HTTP, але не

обмежений цим протоколом. Архітектури типу RESTful можуть бути засновані на інших протоколах прикладного рівня, якщо вони вже реалізують великий і єдиний словник для додатків, заснованих на передачі значущих уявлень станів. Додатки RESTful збільшують використання вже існуючих добре певних інтерфейсів та інших вбудованих можливостей, пропонованих обраним мережевим протоколом, а також скорочують додавання до нього нових можливостей, специфічних для додатка.

RESTful веб-служба (також звана RESTful web API) - це проста веб-служба, реалізована з використанням HTTP і принципів REST.

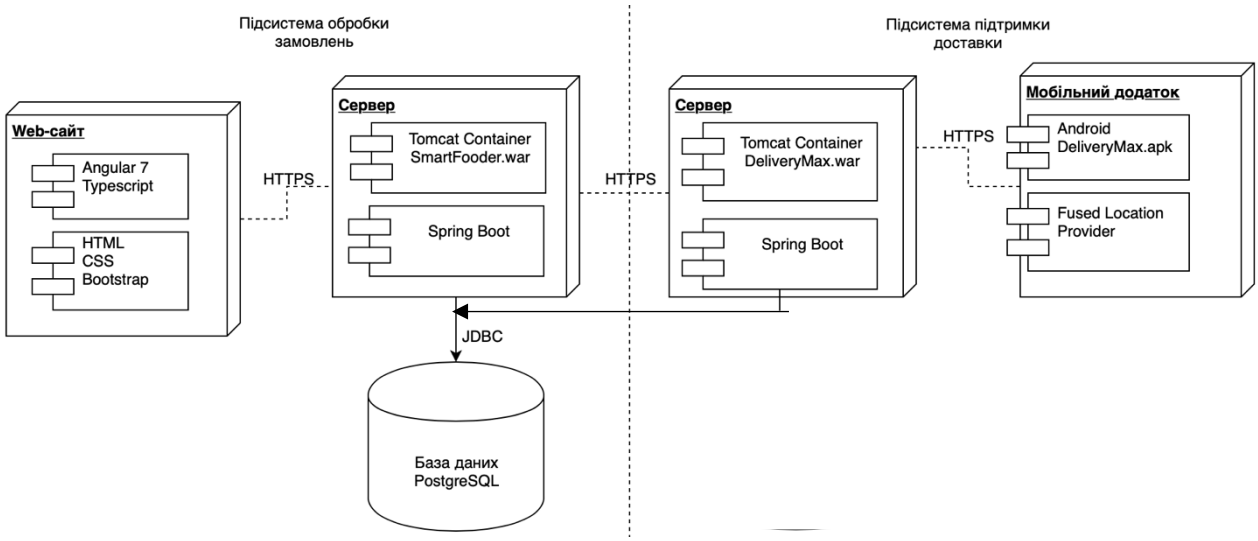

Рисунок 3.11 – Архітектура програмного забезпечення

Додаток обробки замовлень має сервер на базі Tomcat Container, серверна логіка написана за мові програмування Java із застосуванням сучасного фреймворку Spring Boot, який складається із декілька маленьких модулів, один з яких активно використовується в системі це Spring Data JPA. Цей модуль дозволяє нам взаємодіяти із базою даних, це додатковий шар абстракції над JDBC, який вже використовується для взаємодії із базою даних. Підключення до бази даних відбувається у час запуску серверу. Для забезпечення контролю

над змінами у базі даних використовується система управління версіями бази даних LiquiBase.

Веб-сайт обробки замовлень написаний на Angular 7, метою якого є створення комфортного способу взаємодії із підсистемою клієнтами.

Сам веб-сайт напряму ніяк не взаємодіє із базою даних, все відбувається через запити до серверу, який вже уміє взаємодіяти із СУБД. Усі запити відправляється через протокол HTTP із використанням REST підходу до архітектури мережевих протоколів. Дані відправляються у вигляди JSON об’єктів. Такий підхід зручний тим, що у майбутньому буде легше замінити чи розширити клієнтську частину підсистеми.

Кожен компонет клієнт-серверної архітектури має свої функції у системі, найголовніші з яких наведені далі.

Сервер-додатку підсистеми призначення водіїв відповідальний за:

– знаходження найоптимальніших водіїв;
– роботу з БД для зберігання даних;
– взаємодію з сервером підсистеми з обробки замовлень;
– взаємодію с Google Cloud та Firebase Cloud.

Мобільний додаток (клієнт) відповідальний за:

– відображення даних з серверу;
– відправку геолокаційних даних на сервер;
– взаємодію з мапами;
– надання зручного інтерфейсу користувачам для взаємодії з підсистемою підтримки доставки.

Діаграма розгортання для підсистеми призначення водіїввикористовується для того, щоб мати графічне представлення інфраструктури, на яку буде розгорнуто програму (див. рис. 3.12).

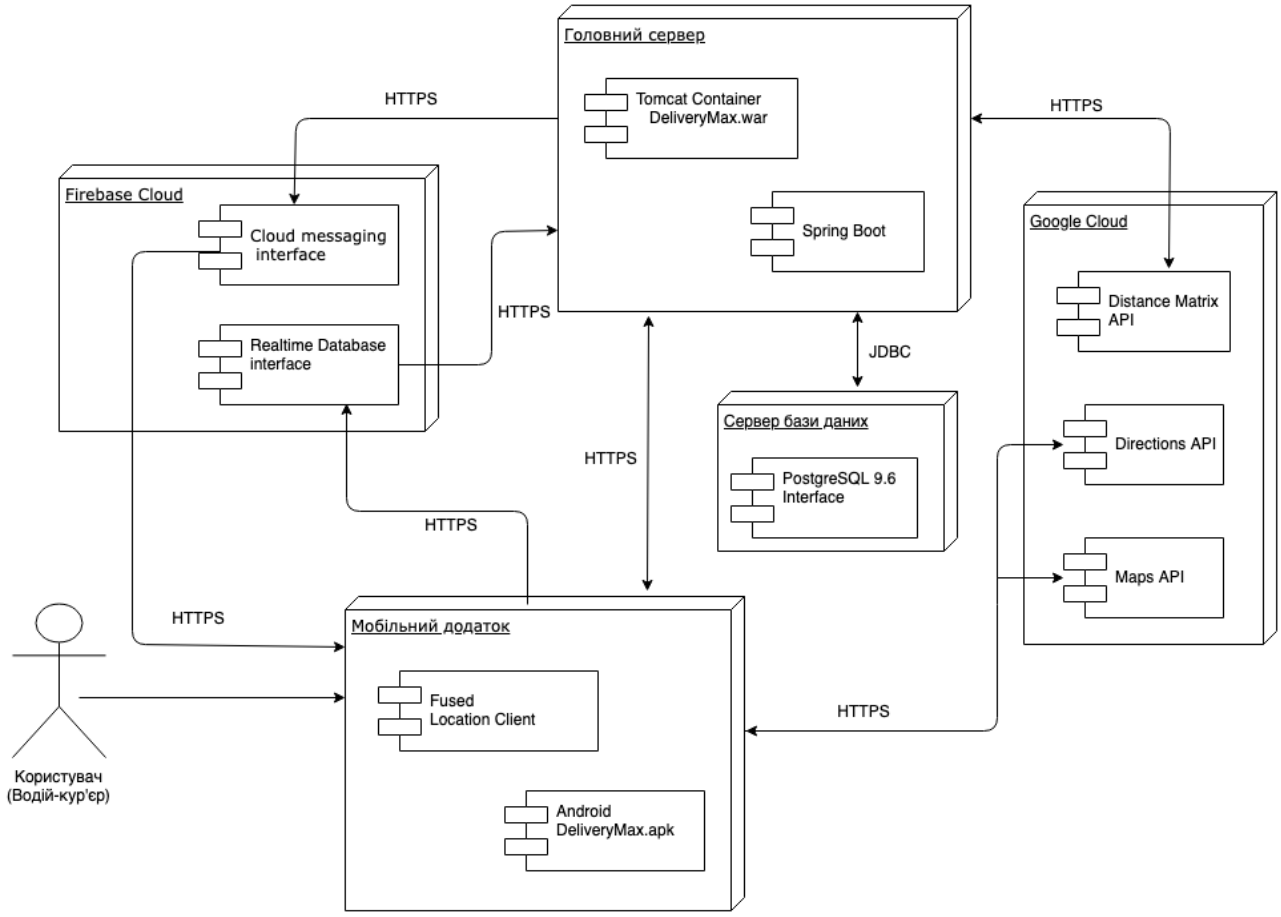

Рисунок 3.12 – Діаграма розгортання для підсистеми призначення водіїв

На базі розробленої архітектури можна приступати до програмної реалізації системи, використовуючи обрані вище технології.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Опис фізичної моделі бази даних

Для зберігання реляційної БД було обрано СУБД PostreSQL 9.6. Для програмних маніпуляцій з даними використовується JPA – Java Persistence API, яке в свою чергу використовує драйвер JDBC – Java Database Connectivity для доступу до даних. Завдяки цьому відпадає необхідність прямої роботи з базою даних, що значно поліпшує процес розробки програмної системи.

Атрибути, які наявні в таблицях БД відповідають наступним умовам:

– усі унікальні ідентифікатори мають числовий тип даних та генеруються засобами СУБД;

– усі текстові атрибути мають максимальний розмір 255 знаків;

– дані про об’єм, координати, вагу та вартість зберігаються у вигляді дробових чисел подвійної точності;

– дата та час зберігаються у числовому форматі, який відповідає кількості пройдених мілісекунд з моменту настання 01.01.1970.

Завдяки бібліотеці Hibernate, яка використовується фреймворком Spring для маппінгу даних з класами програмної системи була створена фізична модель бази даних для підсистеми призначення водіїв, яка наведена на рисунку 4.1.

З фізичної схеми бази даних видно, що таблиця водіїв (drivers) використовує таблицю users та user_authorities. Ці дві таблиці є необхідними для правильної та безпечної роботи серверу. Для того, щоб він міг розпізнати водія, останній повинен володіти певними правами доступу до системи.

Ангалогічно було розроблено фрагмент фізичної моделі бази даних для вирішення задачі про розподіл замовлень.

Розроблена фізична модель БД дозволила зберігати всю необхідню інформацію для поботи серверів-додатку, мобільного та веб-додатків.

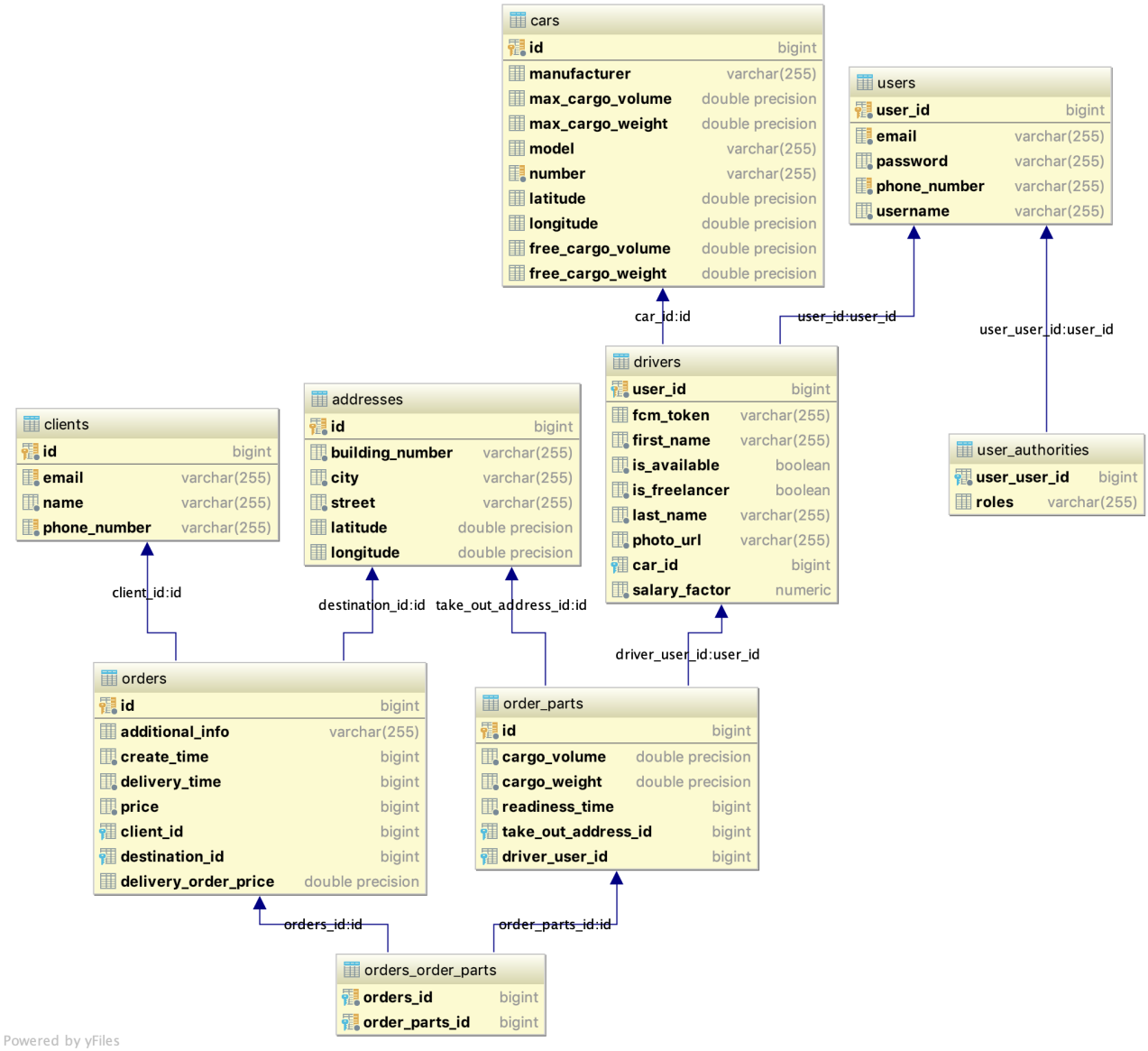

Рисунок 4.1 – Фізична модель бази даних для підсистеми призначення водіїв

4.2 Опис інтерфейсу веб-додатку

Інтерфейс додатку пов’язаний із моделлю системи за допомогою технології Angular 7. Angular представляє фреймворк від компанії Google для створення клієнтських додатків [13]. Перш за все він націлений на розробку SPA-рішень (Single Page Application), тобто односторінкових додатків. В цьому

плані Angular є спадкоємцем іншого фреймворка AngularJS. У той же час Angular це не нова версія AngularJS, а принципово новий фреймворк. Angular надає таку функціональність, як двостороннє зв'язування, що дозволяє динамічно змінювати дані в одному місці інтерфейсу при зміні даних моделі в іншому, шаблони, маршрутизація і так далі.

Для реалізації інтерфейсу сайту було обрано мову розмітки HTML та інструментарій Bootstrap. Bootstrap - це інструментарій з відкритим вихідним кодом для розробки за допомогою HTML, CSS і JS.

Після відкриття сайту користувач бачить вікно для авторизації до системи. З цього вікна користувач після введення своїх даних буде авторизований в залежності його ролі у системі. Всього ролів дві: користувач та адміністратор. Треба зазначити, що адміністратор має усі права користувача, так можливість додавати та редагувати данні системи (див. рис. 4.2).

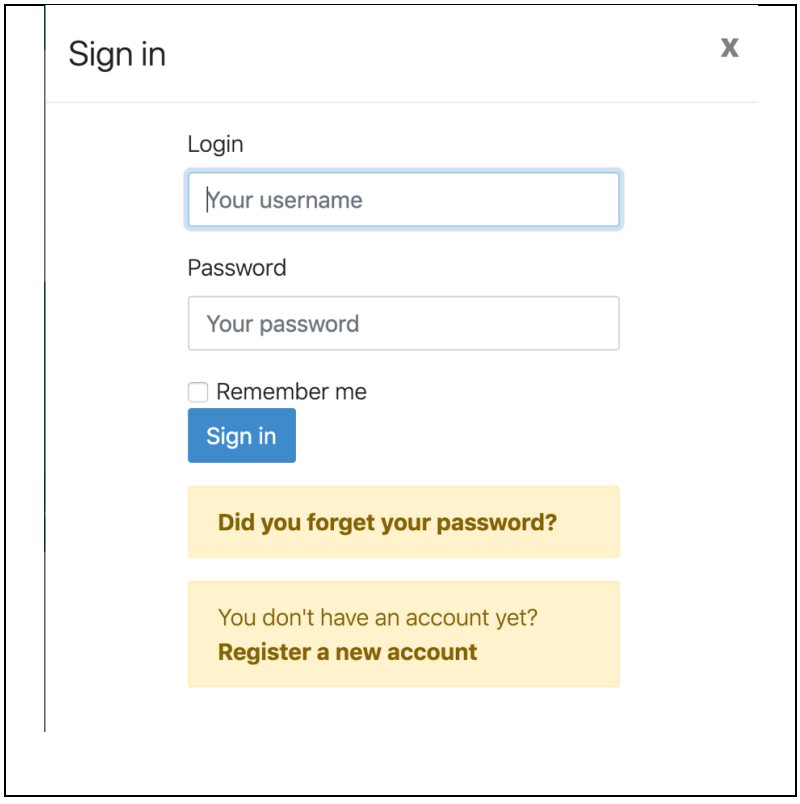

Рисунок 4.2 – Вікно авторизації

Після авторизування користувач сайту бачить список мереж закладів харчування у картковому вигляді, кожен заклад має своє зображення, якщо зображення не було додано адміністратором, показується зображення за замовченням (див. рис. 4.3).

Стиль додатку стандартний для систем такого роду. зверху можна побачити меню, В залежності від користувача система меню дещо відрвзнчється. Звичайний користувач має доступ до кнопок “Home”, “Account” та “Your Order”. Адміністратор додатково бачить ще кнопки “Fill café data”, “Administration”. За кнопкою “Fill café data” адміністратор може побачити усі типи моделей та об’єкти системи, а також може маніпулювати ними. У пункті меню «Administration» можна конфігурувати систему.

На рисунку 4.4 показано типову форму додатку з редагування даних про об’єкти на прикладі об’єктів типу «Заклади харчування». В інтерфейсі надано типові можливості адміністратору з додавання, видалення та редагування кафе.

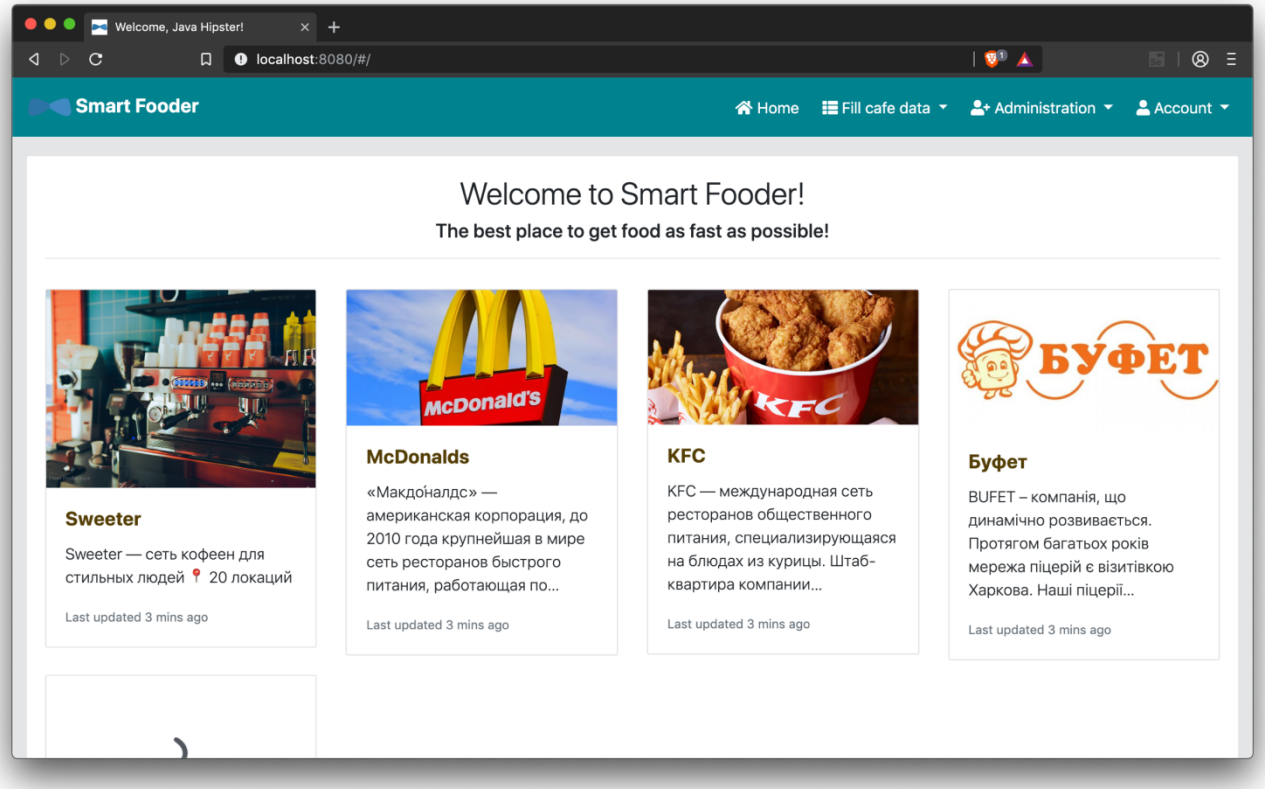

Рисунок 4.3 – Список закладів харчування

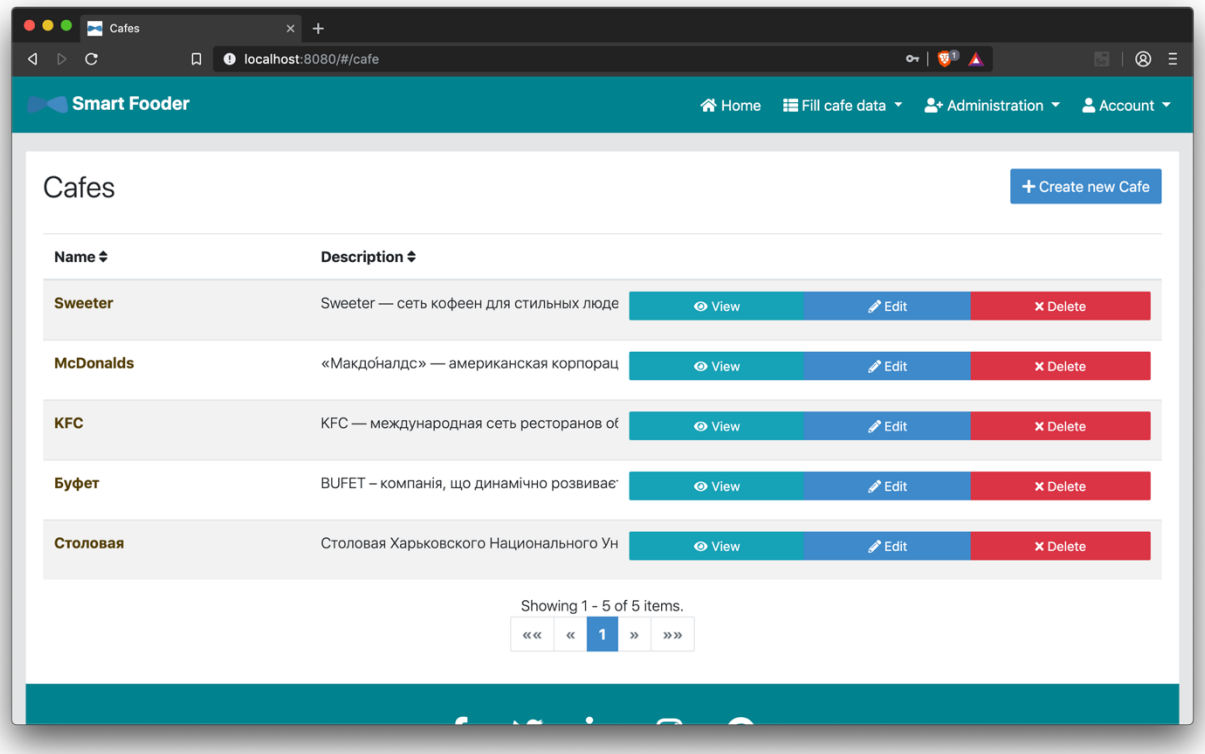

Рисунок 4.4 – Вікно редагування закладів харчування

Після того, як користувач обрав заклад із переліку закладів, він може перейти до вікна Меню, де представлений список усіх страв обраного закладу харчування. Користувач може обирати страви для замовлення. Данні про замовлення зберігаються у сесії браузера користувача, і. до тих пір, поки користувач знаходиться на сайті, система зберігає данні про актуальне замовлення. Користувач може відкрити вікно актуального замовлення та збільшити або зменшити кількість порції. Для цього йому потрібно натиснути на кнопку “Your Order” (див. рис. 4.5).

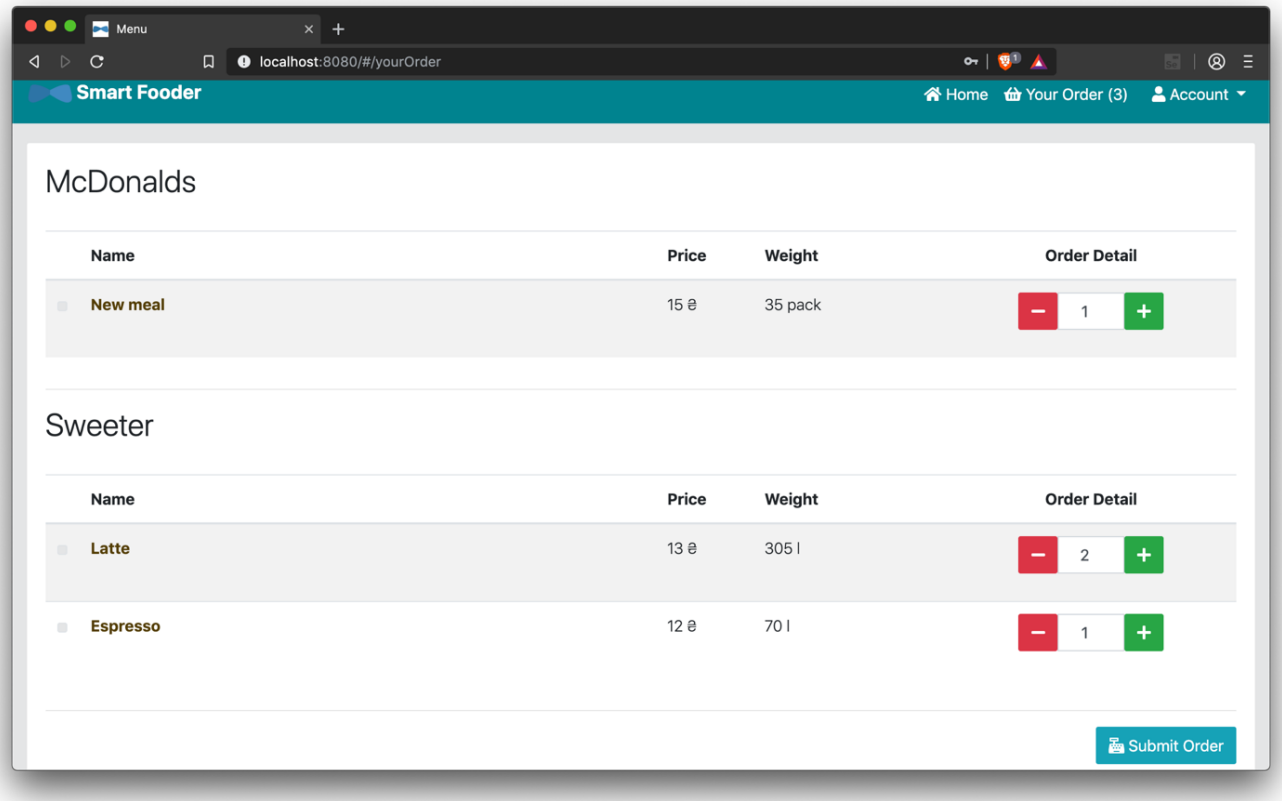

Рисунок 4.5 – Вікно редагування замовлення

Для продовження та оформлення замовлення, користувач повинен натиснути на кнопку “Submit Order”. Потім користувач може ознайомитись із статусом свого замовлення.

Також в системі користувач може змінити свої логін та ім’я за допомогою вікна “Settings”. У вікні “Password” користувач може змінити свій пароль.

Як вже зазначалося, в ролі адміністратора користувач може управляти різними об’єктами даних. Наприклад, він може додавати нові страви в меню. Після натискання назви об’єкта даних “meal”, користувач бачить список усіх страв системи, а зверху кнопку “Create new Meal”, після натискання на яку можна перейде до вікна додавання страви (див. рис. 4.6).

Для зберігання введеної інформації користувач повинен натиснути кнопку “Save”. А коли він перейде назад до списку усіх страв, то в списку з’явиться нова страва.

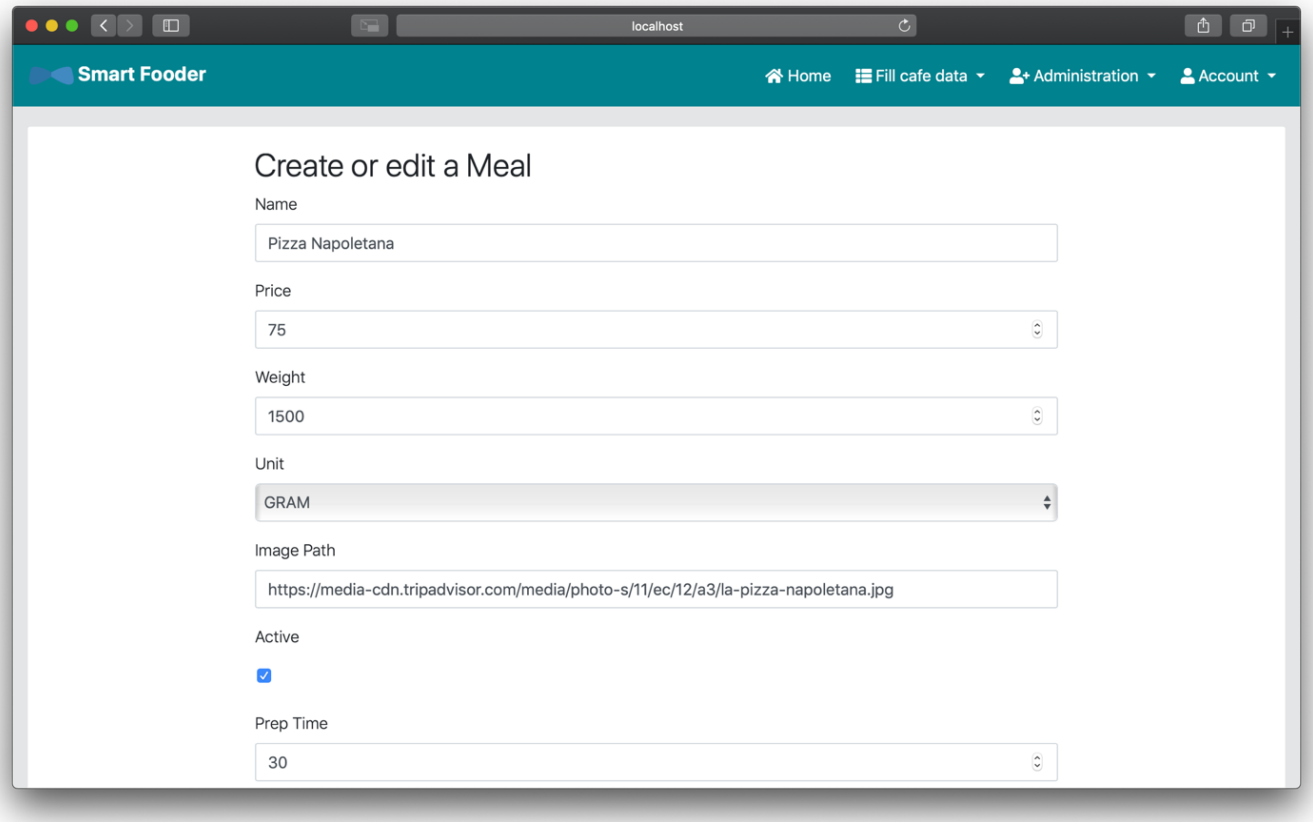

Рисунок 4.6 – Вікно додавання страви

Схожий алгоритм роботи передбачено для усіх сутностей системи, а саме користувач в своїх ролях в веб-додатку може працювати з такими об’єктами, як: заклади харчування, страви, порції, замовлення, інгредієнти, продукти, поставки та постачальники.

4.3 Опис інтерфейсу мобільного додатку

Мобільний додаток для роботи водіїв був розроблений на базі операційної системи Android, тому що вона надає весь необхідний функціонал для роботи з геолокацією та досить популярною. У якості мінімальної підтримуваної версії Android було обрано API 23, тобто Marshmello [14] , з

якою можуть працювати близька 75% смартфонів. Для розробки користувацького інтерфейсу був обран Google Material Design, який дозволяє забезпечити фізичні якості усіх об’єктів як в реальномі житті, наприклад, тінь від кнопки або ефект натискання.

Мобільний додаток створено для водіїв-кур’єрів, отже він повинен надавати наступні функції:

- авторизація та реєстрація у системі;
- отримання миттєвих повідомлень із пропозиціями про доставки;
- прийняття або відхилення пропозиції доставки;
- перегляд списку виконаних доставок;
- детальний перегляд обраної доставки;
- перегляд мапи;
- побудова маршруту за заданими точками на мапі;
- відправка на сервер останніх даних геопозиції.

Під час реалізації додатку були використані промислові стандарти розробки під ОС Android, архітектура має всі властивості Clean Architecture стандарту [15]. Головним правилом такого підходу до архітектури є те, що кожен внутрішній компонент не знає де і як буде використаний зовнішніми компонентами. Розглянемо основні компоненти: UI, Presenters (View Models), Use Cases (Interactors), Entities. UI – це компонент для відображення користувацького інтерфейсу, з ним взаємодіє користувач. Presenters (View Models) – компонент, який виконує логіку та зберігє стан для кожного екрану, з яким взаємодіє користувач. Use Cases (Interactors) представляє собою «логічний двигун» усього додатку. Тобто бізнес логіка Use Cases (Interactors) може бути використана будь-якими екранами. Entities – це бізнес об’єкти, які і складають основу для додатку. Такий підхід до розробки мобільного додатку дозволяє досить легко супроводжувати та маштабувати програмний код за необхідністю.

У якості UI-архітектури було обрано MVVM (Model-View-ViewModel). Model – це бізнес дані для відображення. View – це елементи користувацького

інтерфейсу. ViewModel – це компонент з Android SDK для правильної обробки стану та логіки екрану.

На рисунку 4.7 представлений екран реєстрації водіїв.

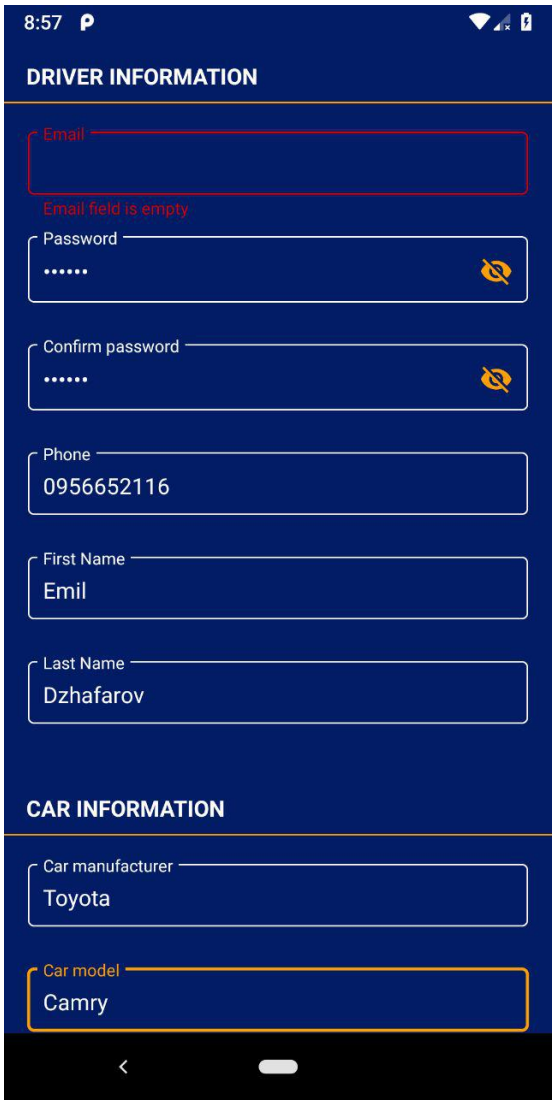

Рисунок 4.7 – Екран реєстрації водіїв

На цьому екрані користувач має можливість увести всі необхідні дані про себе та свій автомобіль. Головний екран додатку представляє собою екран з відкритою мапою (див. рис. 4.8).

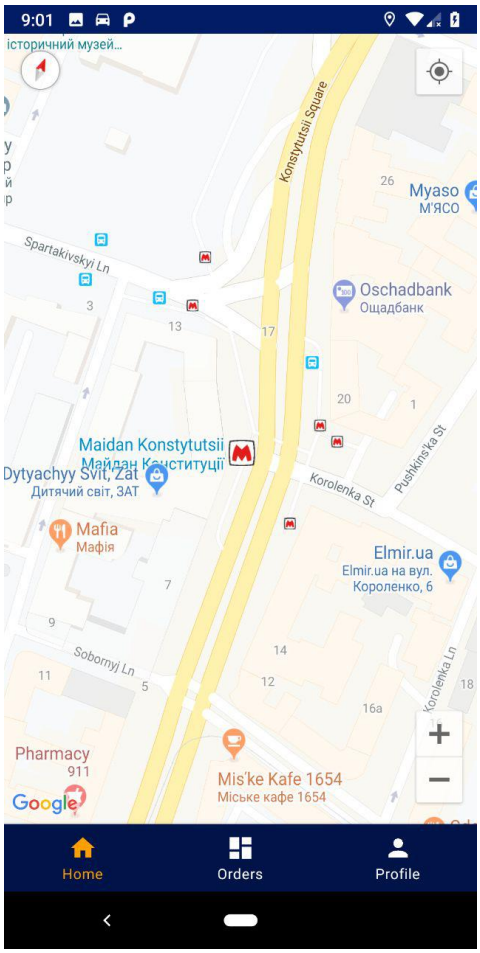

Рисунок 4.8 – Головний екран мобільного додатку

Головна функція додатку є навігація по мапі, тож для цього надано три кнопки знизу. Слід зазначити, що при першому відкритті цього екрану запускається сервіс для збору геолокаційних даних девайсу, попередньо запросивши дозвіл користувача.

Переглядати свої вже виконані замовлення водій може через окремий екран (див. рис. 4.9). На цьому екрані можна бачити список елементів з інформацією про клієнта, пункт призначення, вартість замовлення, час виконання доставки. Для того, щоб побачити більш детальну інформацію про замовлення необхідно перейти на інший екран додатку.

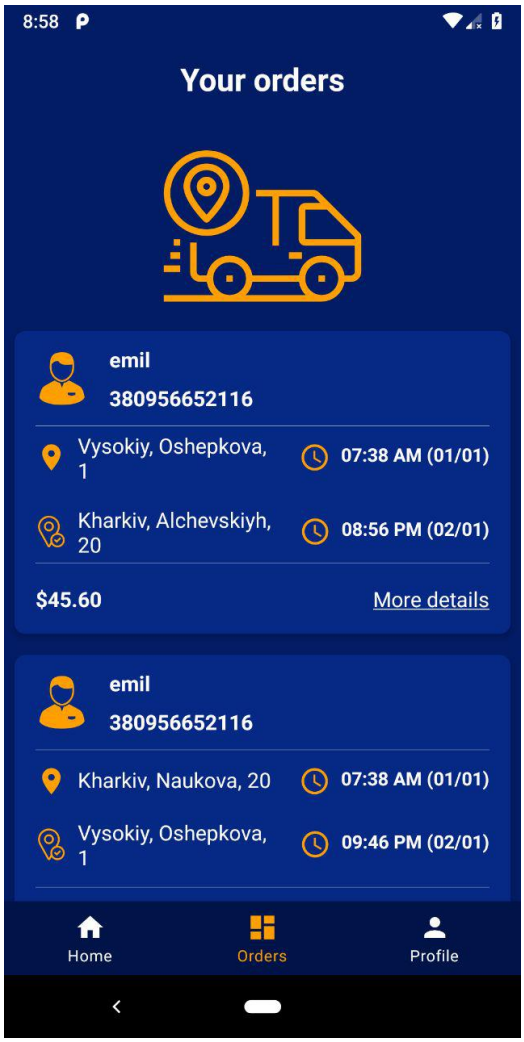

Рисунок 4.9 – Екран перегляду доставок

На рисунку 4.10 показано екран з детальним описом доставки. Як видно з рисунку, водій має можливість підтвердити або скасувати пропозицію про доставку, для чого передбачені відповідні кнопки у нижній частині екрану. Функціонал для підтвердження або скасування пропозиції доставки працює завдяки сервісу Firebase Realtime Database, який використовує NoSQL підхід для організації даних. Цей сервіс дозволяє сповіщувати усіх своїх активних підписників, якщо у БД в конкретному місці сталися зміни.

Мобільний додаток також надає можливість редагування користувацького профілю. Для цього існує вкладка «Profile» на головному екрані. Також водій має можливість змінювати наступні дані про себе та свій

автомобіль: ім’я, прізвище, номер телефону, модель та марка автомобіля, номерний знак, максимальна вантажопідйомність автомобіля (у кг).

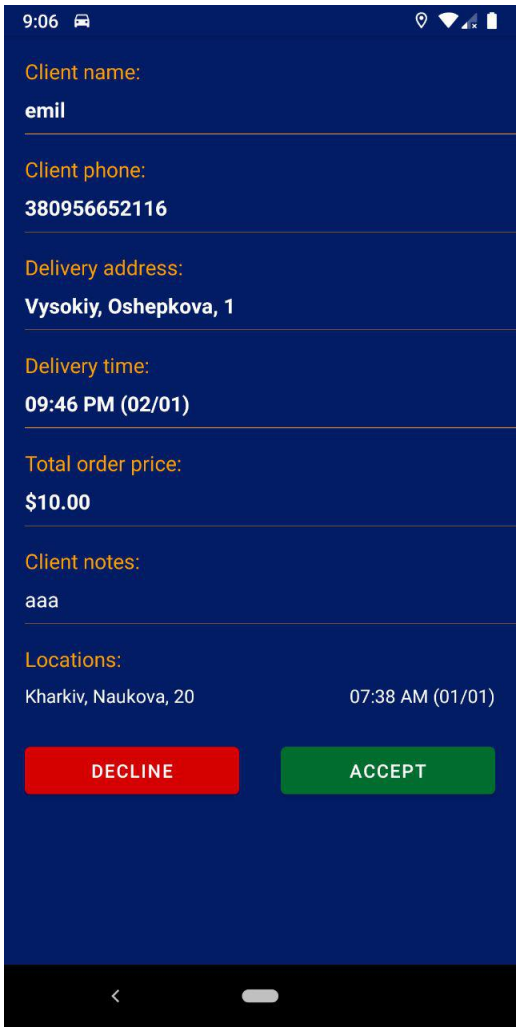

Рисунок 4.10 – Детальний опис доставки

Найважливішою функцією мобільного додатку є перегляд побудованого маршруту на мапі. Він будується одразу як водій прийняв замовлення. Маршрут складається з опорних точок: поточне місцезнаходження водія, адреса для отримання частини замовлення та адреса пункту призначення.

4.4 Опис програмної реалізації алгоритму серверу додатку

Відповідно до розробленої архітектури система має сервер додатку, яки дозволяє реалізувати складні алгоритмі, що задають бізнес-логіку додатку. В нашому випадку бізнес-логіка складається з алгоритмів вирішення оптимізаційних задач:

- задачі розподілу замовлення між ЗХ для приготування;
- задача про призначення водіїв на доставку (див. рис. 4.11).

```
@Component("greedyAlgorithm")
public class GreedyAlgorithm {
    private final int[] radiusZones = { 1_000, 3_000, 5_000, 10_000, 20_000
};

    private final DurationEvaluator durationEvaluator;
    private final OrderPriceEvaluator orderPriceEvaluator;

    @Autowired
    public GreedyAlgorithm(DurationEvaluator durationEvaluator,
OrderPriceEvaluator orderPriceEvaluator) {
        this.durationEvaluator = durationEvaluator;
        this.orderPriceEvaluator = orderPriceEvaluator;
    }

    Driver findPermanentDriver(OrderPart orderPart, List<Driver> allDrivers)
{
        return findAppropriateDriver(orderPart, allDrivers, driver ->
!driver.getIsFreelancer(), 0);
    }
    Driver findAnyDriver(OrderPart orderPart, List<Driver> allDrivers,
List<Long> ids) {
        return findAppropriateDriver(orderPart, allDrivers, driver ->
!ids.contains(driver.getId()), 0);    }
```

Рисунок 4.11 – Програмний код GreedyAlgorithm

```
    private Driver findAppropriateDriver(OrderPart orderPart,
List<Driver> allDrivers, Predicate<Driver> predicate, int position) {
        if (position >= radiusZones.length) {
            return null;
        }

        Location location = orderPart.getTakeOutAddress().toLocation();

        Predicate<Driver> complexPredicate = predicate
                .and(Driver::getIsAvailable)
                .and(driver -> driver.getCar().getFreeCargoVolume() >
orderPart.getCargoVolume() && driver.getCar().getFreeCargoWeight() >
orderPart.getCargoWeight())
                .and(driver ->
LocationUtils.isInsideCircle(radiusZones[position], location,
driver.getCarLocation()))
                .and(driver -> {
                    long duration = durationEvaluator.evaluate(location,
driver.getCarLocation());
                    return duration !=
DurationEvaluator.DURATION_NOT_CALCULATED &&
                            TimeUnit.SECONDS.toMinutes(duration) <=
DurationEvaluator.DRIVER_NOTIFYING_TIME_MINUTES;
                });

        return allDrivers.stream()
                .filter(complexPredicate)
                .min(getDriverComparator(location))
                .orElseGet(() -> findAppropriateDriver(orderPart,
allDrivers, predicate, position + 1));
    }
    private Comparator<Driver> getDriverComparator(Location location) {
        return (driver1, driver2) -> {
            double price1 = orderPriceEvaluator.evaluateForSingleDriver(
                    new LatLng(location.getLatitude(),
location.getLongitude()),
                    new LatLng(driver1.getCar().getLatitude(),
driver1.getCar().getLongitude()),
                    driver1.getSalaryFactor()
            );
            double price2 = orderPriceEvaluator.evaluateForSingleDriver(
                    new LatLng(location.getLatitude(),
location.getLongitude()),
                    new LatLng(driver2.getCar().getLatitude(),
driver2.getCar().getLongitude()),
                    driver2.getSalaryFactor()
            );
            return Double.compare(price1, price2);
        };
    }
}
```

Рисунок 4.11, аркуш 2

Алгоритм пошуку водіїв був розділений на два об’єкти: перший об’єкт має на меті «жадібний» пошук найоптимальнішого водія для заданої частини замовлення – GreedyAlgorithm, а другий об’єкт виконує логіку з обробки замовлення та закріплення водія за частиною замовлення – OrderHandler. Реалізація складова алгоритму GreedyAlgorithm представлена на рисунку 4.11.

Як видно з програмного коду, алгоритм пошуку найоптимальнішого водія використовує рекурсію. Спочатку обирається зона найближчого розташування до заданої адреси радіусом 1000 метрів. Якщо у колі цього радіусу не буде знайдено водія, тоді радіус буде збільшений до наступного – до 3000 метрів і т.д.

Якщо було знайдено декілька водіїв в одній зоні, то обирається той, який є найбільш економічно ефективним. Тобто з обраних водіїв вибирається той, вартість поїздки якого буде найменшою з урахуванням його зарплатного коефіцієнту. Зарплатний коефіцієнт у штатного водія складає 1, в той час як у фрілансера – 0.75.

Так наприклад, якщо водій-фрілансер знаходиться на відстані в 2000 метрів від заданої точки, а штатний водій знаходиться на відстані 1800 метрів від тієї ж точки, то буде обраний саме водій фрілансер, адже: $(P + P * 0.75) * 2000 < (P + P * 1) * 1800$, де P – це єдина вартість поїздки на 1000 метрів.

Повна реалізація алгоритму пошуку водіїв представлена у додатку Б.

5 АНАЛІЗ МОЖЛИВИХ ЗАСТОСУВАНЬ ТА ТЕСТУВАННЯ ПРОГРАМНОІ СИСТЕМИ

5.1 Аналіз можливих застосувань програмної системи

Доставка їжі вже давно стала частиної життя сучасної людини. Швидкі та сучасні сервіси доставки їжі та напоїв заполонили наше життя. Мільйони людей замовляють їжу додому та на роботу. Отже, дуже важливо створити якісний сервіс, що буде заощаджувати час та гроші користувачів.

Незважаючи на те, що існує багато сервісів з доставки їжі та напоїв, в мережах закладів харчування все ще виникати складності, що пов’язані з бізнес-логікою, наприклад, обслуговування великих замовленнями, оптимізацією доставки замовлень та інше.

Програмний продукт який створено вирішує ці складності та реалізвє наступні функції:

- реєстрація та авторизація користувачів та водіїв;
- додавання, редагування та видалення необхідної інформації стосовно закладів харчування, їх меню, замовлень, тощо;
- оформлення замовлення за обраними стравами користувачем;
- підтримка основних бізнес-процесів з замовлення та доставки їжі, наприклад, оптимізація замовлення системою шляхом розподілу його для приготування між декількома закладами харчування;
- відправка повідомлень водіям про доставку, прийняття або відхилення пропозиції доставки для водіїв-фрілансерів;
- редагування персональної інформації у мобільному додатку та інше.

Дана система, таким чином, може знайти своє застосування в мережах онлайн-закладів харчування, які хочуть зайняти великий сегмент ринку та забезпечити доставку їжі та напоїв як в малих, так і в великих обсягах.

Між тим, в даній реалізації системи ще є напрямки, за якими система може бути вдосконалена, а саме:

- розширення умов, що можуть бути враховані в оптимізаційній задачі про розподіл великих замовлень; наприклад, додавання умов про врахування наявності продуктів на складах ЗХ;
- розширення можливостей з організації доставки замовлень, а саме додати до моделі можливість призначати одного водія на декілька доставок;
- реалізація це одного клієнтського додатку для підтримки роботи самих ЗХ для реалізації таких бізнес-процесів як постачання продуктів, облік з приготування страв та інше.

5.2 Опис тестування розробленої програмної системи

Всі види тестування програмного забезпечення, в залежності від переслідуваних цілей, можна умовно розділити на наступні групи [14]:

а) функціональні;

б) нефункціональні;

в) пов'язані зі змінами.

Функціональні тести базуються на функціях і особливостях, а також взаємодії з іншими системами, і можуть бути представлені на всіх рівнях тестування: компонентному або модульному (Component / Unit testing), інтеграційному (Integration testing), системному (System testing) і приймальному (Acceptance testing). Функціональні види тестування розглядають зовнішню поведінку системи. Далі перераховані одні з найпоширеніших видів функціональних тестів:

а) функціональне тестування (Functional testing);

б) тестування безпеки (Security and Access Control Testing);

в) тестування взаємодії (Interoperability Testing).

Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними

величинами. В цілому, це тестування того, "Як" система працює. Далі перераховані основні види не функціональних тестів:

а) тестування навантаження (Performance and Load Testing);

б) стресове тестування (Stress Testing);

в) тестування стабільності або надійності (Stability / Reliability Testing);

г) об'ємне тестування (Volume Testing);

д) тестування установки (Installation testing);

е) тестування зручності користування (Usability Testing);

є) тестування на відмову і відновлення (Failover and Recovery Testing);

ж) конфігураційне тестування (Configuration Testing).

Для обраного програмного забезпечення було вирішено використовувати функціональне тестування, тестування взаємодії та стрес-тестування.

Функціональне тестування розглядає заздалегідь вказане поведінку і грунтується на аналізі специфікацій функціональності компонента або системи в цілому.

Функціональні тести ґрунтуються на функціях, виконуваних системою, і можуть проводитися на всіх рівнях тестування (компонентному, інтеграційному, системному, приймальному). Як правило, ці функції описуються в вимогах, функціональних специфікаціях або у вигляді випадків використання системи (use cases).

Переваги функціонального тестування:

а) імітує фактичне використання системи.

Недоліки функціонального тестування:

а) можливість упущення логічних помилок в програмному забезпеченні;

б) ймовірність надмірного тестування.

З розвитком мережевих технологій та інтернету взаємодія різних систем, сервісів і додатків один з одним набуло значної актуальності, так як будь-які пов'язані з цим проблеми можуть привести до падіння авторитету компанії, що як наслідок спричинить за собою фінансові втрати. Тому до тестування взаємодії варто підходити з усією серйозністю.

Тестування взаємодії (Interoperability Testing) - це функціональне тестування, що перевіряє здатність додатки взаємодіяти з одним і більше компонентами або системами і включає в себе тестування сумісності (compatibility testing) і інтеграційне тестування (integration testing).
Програмне забезпечення з хорошими характеристиками взаємодії може бути легко інтегрована з іншими системами, не вимагаючи будь-яких серйозних модифікацій. В цьому випадку, кількість змін і час, необхідний на їх виконання, можуть бути використані для вимірювання можливості взаємодії.

Під час дипломного проектування були написані автоматизовані інтеграційні тести для серверної частини підсистеми, ці тести покривають майже усі контролери та їх основні кінцеві точки.

Інтеграційне тестування - вид тестування, при якому на відповідність вимог перевіряється інтеграція модулів, їх взаємодія між собою, а також інтеграція підсистем в одну загальну систему. Для інтеграційного тестування використовуються компоненти, вже перевірені за допомогою модульного тестування, які групуються в безлічі. Дані безлічі перевіряються відповідно до плану тестування, складеним для них, а об'єднуються вони через свої інтерфейси.

Тести були написано із допомогою JUnit та Spring MVC Testing інструментами. JUnit - бібліотека для модульного тестування програм Java. Створений Кентом Беком і Еріком Гамою, JUnit належить родині фреймворків xUnit для різних мов програмування, що бере початок в SUnit Кента Бека для Smalltalk. JUnit породив екосистему розширень - JMock, EasyMock, DbUnit, HttpUnit і т. д [15].

Так як модулі з'єднуються між собою за допомогою передбачених реалізацією інтерфейсів і в процесі тестування у нас немає потреби розглядати внутрішню структуру компонентів, можна стверджувати, що інтеграційне тестування виконується методом «чорного ящика».

Виходячи з відмінностей між модульним тестуванням і системним тестуванням, інтеграційне тестування є перехідним етапом між поданням програми у вигляді окремих модулів в вид повністю функціональної системи.

Підтримка юніт та інтеграційного тестування забезпечується у формі анотації Spring SpringContext Framework. Каркас TestContext є незалежним від фактичної тестової системи, що використовується, що дозволяє проводити тестування в різних середовищах, включаючи JUnit, TestNG та інші.

Розроблені тестові випадки ґрунтуються на задачах повсякденного використання інформаційної системи. Всі три тест-кейси успішно виконалися декілька разів, що свідчить про надійність та стабільність програмної системи.

ВИСНОВКИ

Під час атестаційної магістерської роботи було досліджено задачі лінійного програмування з метою побудови бізнес-логіки системи підтримки мережі онлайн-закладів харчування.

Було проведено дослідження бізнес-процесів, що пов’язані з проботою закладів харчування, що спеціалізуються з доставки їжі та напоїв. Було виділено два бізнес-процеси, які потребували алгоритмічної підтримки на базі оптимізаційних задач лінійного програмування:

- процес обробки великих замовлень та розподілу їх для приготування між декількома закладами;
- процес призначення водіїв на доставку замовлень.

Для даних бізнес-процесів було розроблено математичні моделі оптимізаційних задач про розподіл ресурсів та про призначення, відповідно. Для зберігання даних, необхідних для роботи клієнтсткіх додатків, та роботи оптимізаційних моделей було спроектовано базу даних. Розроблено алгоритмі вирішення оптимізаційних задач на базі жадібного алгоритму, що дозволяє в режимі реального часу знаходити оптимальне рішення поставлених задач. Розроблені моделі та алгоритми реалізовані під час програмної реалізації системи підтримки роботи мережі онлайн закладів харчування.

Розроблена програмна система реалізовує всі функції, які були визначені під час постановки завдання, а саме: огляд меню закладів харчування; замовлення страв користувачем; обробка замовлень оператором мережі закладів; розподілення великих замовлень для приготування між декількома закладами; реєстрація / авторизація водіїв в системі; пошук найоптимальнішого водія; відправка повідомлень водіям; редагування персональної інформації у мобільному додатку; перегляд попередніх виконаних доставок у мобільному додатку; перегляд детальної інформації про замовлення; отримання миттєвих повідомлень із пропозиціями виконання доставки у мобільному додатку.

Створена у процесі виконання роботи програмна підсистема підтримки доставки є неабияк корисною для мережі онлайн-кафе. Адже вона дозволяє зменшити витрати на доставку великих замовлень їжі завдяки постійному залученню водіїв до роботи, що зменшує час марного їх простою, а отже значно заощаджує кошти. Клієнти у свою чергу отримують можливість замовлення великих обсягів їжі у найкоротший час та за найменшою вартістю доставки.

В ході атестаційної роботи магістра було:

– проведено аналіз бізнес-логіки та моделювання предметної області роботи мережі онлайн-закладів харчування;

– проведено досліждення задач лінійного програмування та методів їх вирішення, обрано найкращі для моделювання основних процесів бізнес-логіки мережі онлайн-закладів харчування;

– розроблено математичні моделі ЗЛП для виявлених основних процесів бізнес-логіки;

– розроблено схему бази даних для збереження інформації предметної області та розроблених математичних моделей;

– розролено алгоритми вирішення промодельованих ЗЛП;

– розроблено архітектуру та програмно реалізовано систему підтримки роботи онлайн-закладів харчування.

За результатами атестаційної роботи магістра було розроблено презентацію (див. додаток А).

Було зроблено подано тези доповіді на Міжнародний молодіжний форум. Тези доповіді наведено в додатку Б. Лістинг коду наведено в додатку В.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Інтернет-видання MC Today: Пообедать, не выходя из офиса. Топ-5 киевских сервисов доставки еды. - URL: https://mc.today/poobedat-ne-vyhodya-iz-ofisa-top-5-kievskih-servisov-dostavki-edy/ (дата звернення 29.05.2019).

2. Что такое Glovo / Интернет издательство Gloss – URL: https://gloss.ua/lifestyle/123692-v-kieve-poyavilos-prilozhenie-dlya-dostavki-produktov-glovo-ob-yasnyaem-chto-eto-i-zachem-ono-vam (дата звернення: 20.03.2019).

3. Джон Шрайбфедер Эффективное управление запасами / Пер. с англ. — 2-е изд. — М.: Альпина Бизнес Букс, 2006. — 304 с.

4. Гребеннік І. В. Методи підтримки прийняття рішень : навч. посібник / І. В. Гребеннік, Т. Є. Романова, А. Д. Тевяшев, Г. М Яськов ; МОН України, Харк. нац. ун-т радіоелектроніки. – Харків : ХНУРЕ, 2010. – 127 с.

5. L.Vlasenko, A. Chikrii On a differential game in a system with distributed parameters // Proceedings of the Steklov Institute of Mathematics, 2016, vol. 292 (1), P. 276-285 URL: https://link.springer.com/article/10.1134/S0081543816020243 (дата звернення: 20.01.2020).

6. Вся высшая математика. Том 6. Вариационное исчисление, линейное программирование, вычислительная математика, теория сплайнов. - М.: Либроком, 2013. - 256 с.

7. Васильев Ф. П., Иваницкий А.Ю. Линейное программирование. - М.: Факториал Пресс, 2016. - 352 с.

8. Wikipedia: Жадный алгоритм, условия применимости, примеры. - URL: https://wikipedia.org/wiki/Жадный_алгоритм (дата звернення: 07.02.2020).

9. Фаулер М., Рейс Д. Шаблоны корпоративных приложений. – М.: Диалектика, 2016.- 544 с.

10. Кренке, Д. Теорія і практика побудови баз даних (8-е вид.). – СПб.: Пітер, 2003. – 800 с.

11. Грабер М. SQL. – К: ЛОРІ, 2003. – 644 с.

12. Wikipedia: Нормалізація баз даних. - URL: https://uk.wikipedia.org/wiki/Нормалізація_баз_даних/ (дата звернення 10.10.2019).

13. Введение в Рефакторинг / Рефакторинг.Гуру – URL: https://refactoring.guru/ru/refactoring (дата звернення: 05.11.2019)

14. Штрауб Б. Git для профессионального программиста. – М.: Вільямс, 2016.- 496 с.

15. Документація з Android архітектури: Використання Android Components. - URL: https://developer.android.com/topic/libraries/architecture/ (дата звернення 20.02.2020).

16. Что такое Uber Eats / Ain.ua Интернет-бизнес в Украине – URL: https://ain.ua/2019/02/06/uber-eats-chto-eto/ (дата звернення: 21.12.2019).

17. Документація з Spring Framework: Початок роботи. - URL: https://spring.io/ (дата звернення 10.03.2020).

18. Angular Docs / Angular Framework – URL: https://angular.io/docs – (дата звернення: 05.03.2020)

19. PostgreSQL 9.6.13 Documentation / PostgreSQL: Documentation – URL: https://www.postgresql.org/docs/9.6/index.html (дата звернення: 14.11.2019).

20. Мацяшек Л. Анализ и проектирование информационных систем с помощью UML 2.0. – М.: Вільямс, 2016.- 816 с.

21. Бородакий Ю.В. Линейное программирование в современных задачах оптимизации. - М.: МИФИ, 2008. - 564с.

22. Гвоздинський А. М. Оптимізаційні задачі в організаційному управлінні. У 2-х частинах. Навч. посібник. - Харків: ХТУРЕ, 1997 – 216 с.

23. William J. Cook, William H. Cunningham, William R. Pulleyblank, Alexander Schrijver, Combinatorial Optimization 1st Edition, 287 с. — 1998.

24. Документація з Hibernate: Початок роботи. - URL: https://hibernate.org/ (дата звернення 10.04.2020).

25. Інтернет-видання HackerNoon: Android Clean Architecture with Kotlin, RxJava and Dagger 2. - URL: https://hackernoon.com/android-clean-architecture-with-kotlin-rxjava-and-dagger-2-6006be2d0c02/ (дата звернення 20.01.2020).

26. Рэшка Джефф, Дастин Элфрид, Пол Джон. Тестирование программного обеспечения. - Лори, 2012. - 568 с.

27. Виды Тестирования Программного Обеспечения / Тестирование ПроТестинг – URL: http://www.protesting.ru/testing/testtypes.html – (дата звернення: 05.04.2020).