Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мережних технологій факультету інформатики

РОЗРОБКА ВЕБ-СЕРВІСУ ДЛЯ ОРГАНІЗАЦІЇ ДОСТАВКИ ЗАМОВЛЕНЬ

Текстова частина до курсової роботи

за спеціальністю „Комп'ютерні науки та інформаційні технології" 122

Керівник курсової роботи

Канд.фіз.-матем. наук

Гречко А.В.

(прізвище та ініціали)

_____

(підпис)

"____" _____ 2020 р.

Виконала студентка 3 курсу

Печура М.В.

(прізвище та ініціали)

"____" _____ 2020 р.

Київ 2020

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мережних технологій

ЗАТВЕРДЖУЮ
Зав. Кафедри
Малашонок Геннадій Іванович
_____
*(підпис)*_____
“____” _____ 2019 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

Студентки Печури Мілени Володимирівни факультету інформатики 3 курсу
ТЕМА Розробка веб-сервісу для організації доставки замовлень

Вихідні дані: Перегляд магазинів та їх товарів, можливість оформити замовлення, створити та змінити акаунт.
Зміст ТЧ до курсової роботи:
Календарний план
Вступ
Частина 1: Аналіз предметної області. Постановка завдання курсової роботи
Частина 2: Теоретичні відомості про розробку веб-застосувань
Частина 3: Опис реалізації програмного продукту
Висновки
Список використаної літератури
Додатки

Дата видачі “____” ____ 2019 р.
Керівник_____ Завдання отримала _____
(підпис) *(підпис)*

Тема: Розробка веб-сервісу для організації доставки замовлень
Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	8.10.2019	
2.	Пошук тематичної літератури	8.11.2019	
3.	Ознайомлення з літературою	11.12.2019	
4.	Вивчення аналогів	25.12.2019	
5.	Планування веб-сайту	13.01.2020	
6.	Створення бази даних	20.01.2020	
7.	Створення директорій проекту	24.01.2020	
8.	Реалізація створення користувача, сесії та авторизації	29.01.2020	
9.	Створення базової навігації в додатку	07.02.2020	
10.	Розробка фунціоналу головної сторінки	18.03.2020	
12.	Додавання стилів до сторінок	07.04.2020	
13.	Написання текстової частини	20.04.2020	
14.	Подання першої версії записки науковому керівнику	04.04.2020	
15.	Перегляд змісту роботи керівником	06.04.2020	
16.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	10.04.2020	
16.	Створення презентації	11.04.2020	
18.	Захист роботи	20.04.2020	

Студентка Печура М.В.

Керівник Гречко А.В.

“____” _____

ЗМІСТ

Перелік умовних позначень:

1. Магазин – означає організацію, звідки клієнт зробив замовлення: ресторан/аптека/супермаркет/кав’ярня і т.д.
2. Замовник – клієнт. Позначає одну й ту саму роль.
3. Етапи доставки – у якому статусі знаходиться замовлення: кур’єр приїхав у магазин/отримав замовлення і т.д.
4. СКБД – система керування базами даних.
5. Unit-testing - рівень тестування програмного забезпечення, на якому тестуються окремі одиниці/компоненти програмного забезпечення.
6. AI – Artificial Intelligence.
7. UI/UX – User interface/User experience – поняття, які використовуються для опису дизайну та функціональності сайту.
8. HTTP-протокол (Hyper Text Transfer Protocol) – протокол передачі гіпер-текстових документів.
9. БД – база даних.

ВСТУП

В сучасному світі всі задачі максимально оптимізуються і спрощуються, що стосується зокрема і споживання їжі, обирання подарунку на свято, відвідування супермаркету чи аптеки. Замість того, щоб приготувати вдома або сходити в ресторан, люди замовляють доставку страв. В офіс на обід або додому на вечерю. Замість витрачання часу на поїздку до магазину, люди замовляють доставку продуктів, книжок, ліків та канцелярії. На ринку існують декілька служб доставки, але це не заважає компаніям випускати нові продукти-конкуренти. Тому дослідження роботи сервісів доставки та розробка аналогічного є корисним для кожного веб-розробника.

Метою є створення веб-сервісу для організації доставки замовлень. Він полегшить роботу кур'єрів та адміністраторів сайту, а також зробить процес замовлення товару зручним. Кур'єри швидше будуть доставляти, адміністратори – керувати внутрішніми процесами, користувачі – оформлювати і отримувати товар.

Завданням є розробка застосування, яке дасть користувачам можливість замовляти доставку, автоматизувати процес створення замовлення, дозволить з мінімальними зусиллями змінювати деталі доставки, наприклад адресу або спосіб оплати.

Об'єктом дослідження є створення веб-сервісу для організації доставки замовлень.

Предметом дослідження є огляд існуючих сайтів та додатків для доставки: для замовників та для кур'єрів.

Розроблений сервіс є підґрунтям для створення високонавантажених складних систем, які мають користувачів із різними функціями у системі.

Веб-застосунок розроблений на мові Python за допомогою фреймворку Django. Створення та робота з базою даних відбувалась із СКБД PostgreSQL, тобто пакетом програм для запуску сервера, командного рядка для взаємодії з

базою даних. Контроль проекту та написання коду здійснювалось у середовищі PyCharm Professional.

Робота складається зі вступу, трьох основних розділів, кожен з яких містить підрозділи, висновків, списку використаної літератури та додатків.

У першому розділі проаналізовано тему як актуальне питання для дослідження, розглянуто аналоги розробки, які наразі користуються попитом, виявлено їхні недоліки. Сформульовано постановку задачі.

Другий розділ висвітлює теоретичні відомості про розробку веб-застосувань на мові Python, їх структуру. Наведено короткий опис архітектурного шаблону, за яким реалізовано додаток з використанням фреймворку. Також наведено класифікацію сайтів та визначено належність розроблюваного застосунку до одного з типів.

У третьому розділі описується реалізація проекту. Визначені вимоги до даних, які представлені у застосунку, схематично зображені інтерфейси, наведені деталі процесу розробки, такі як написані функції, класи. Значна увага приділена базі даних, яка складається з багатьох сутностей, кожна з яких грає важливу роль у роботі програми. Також обґрунтовано вибір технологій розробки, середовища програмування та мови. Продемонстровано результати тестування програми та описано як воно здійснювалось.

РОЗДІЛ 1

1.1. Аналіз сучасного стану питання та обґрунтування теми

У 21 сторіччі все більше і більше людей користуються сервісами доставки, більше того: мають певні вимоги до них, стають більш прискіпливими, помічають недопрацьовані частини програми. Користувачі вимагають великої кількості функцій, що спростять їм пошук необхідного товару для доставки та мінімізують живий контакт (із кур'єром, працівником магазину). Більшість сервісів надають мінімальний функціонал, в той час як користувач, звісно, вимагає більшого.

Сервіси доставки є одними із найактуальніших у сучасному світі: цікавих для дослідження, розробки та вдосконалення. Вони є популярними та дійсно вкрай необхідними. Тому мій вибір пав на них.

1.2. Огляд існуючих аналогів розробки

Лідерами на ринку доставки [1] є Glovo, UberEats та Raketa, серед яких лише перший сервіс доставляє «все що завгодно» - цитата з головної сторінки їхнього сайту(див.рис.1.2.1).

За словами Дарії Янченко [2], журналістки Retailers, недоліки **Glovo** є такими: 1) незрозуміло що означає ціна за продукт: за кілограм, упаковку чи штуку; 2) сума, яку списали з картки, була на гривню більше, ніж вказана у чеку. А ведучий блогу OH MY GADGET [3] показав у своєму відео, що існують баги у секції «Найпопулярніше у Вашому місці»(див.рис.1.2.2) головної сторінки, а саме при натисканні на деякі магазини відображається сторінка з помилкою, в той час якщо ввести назву цього магазину в Пошуку, то при натисканні відобразиться меню магазину. Проте такі баги лише на сайті, адже у додатку такої секції немає. Також ведучому не було зрозуміло, що секція замовлення у додатку знаходиться внизу екрану у вигляді логотипу Глово.

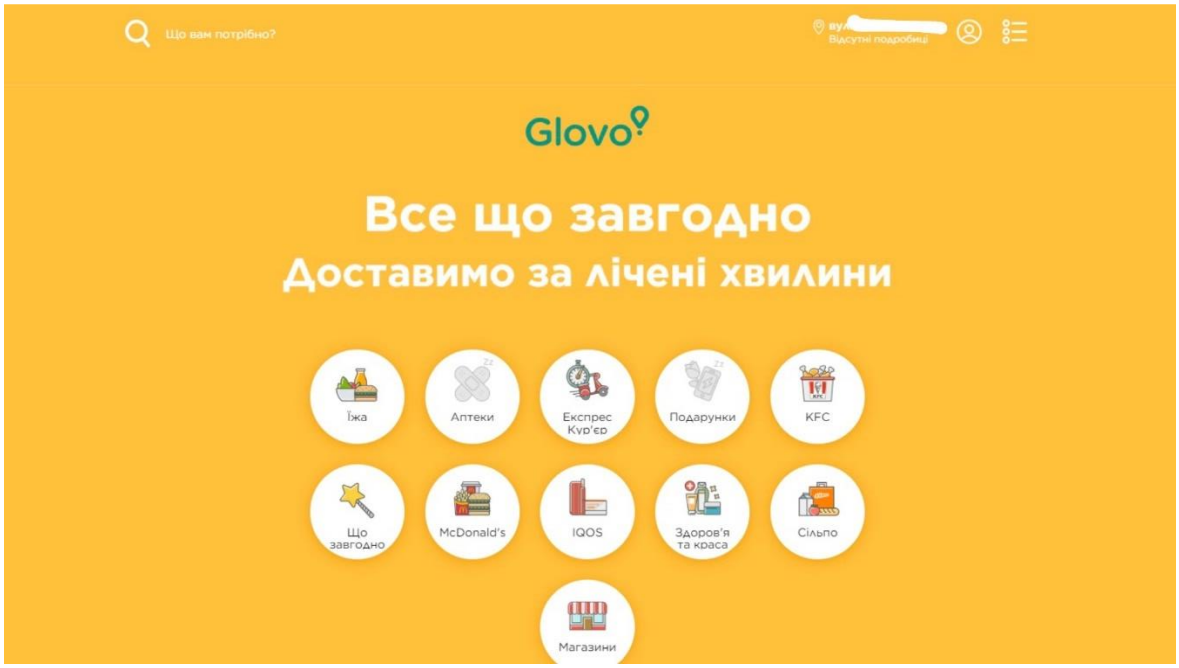


Рис. 1.2.1. Головна сторінка сервісу Glovo

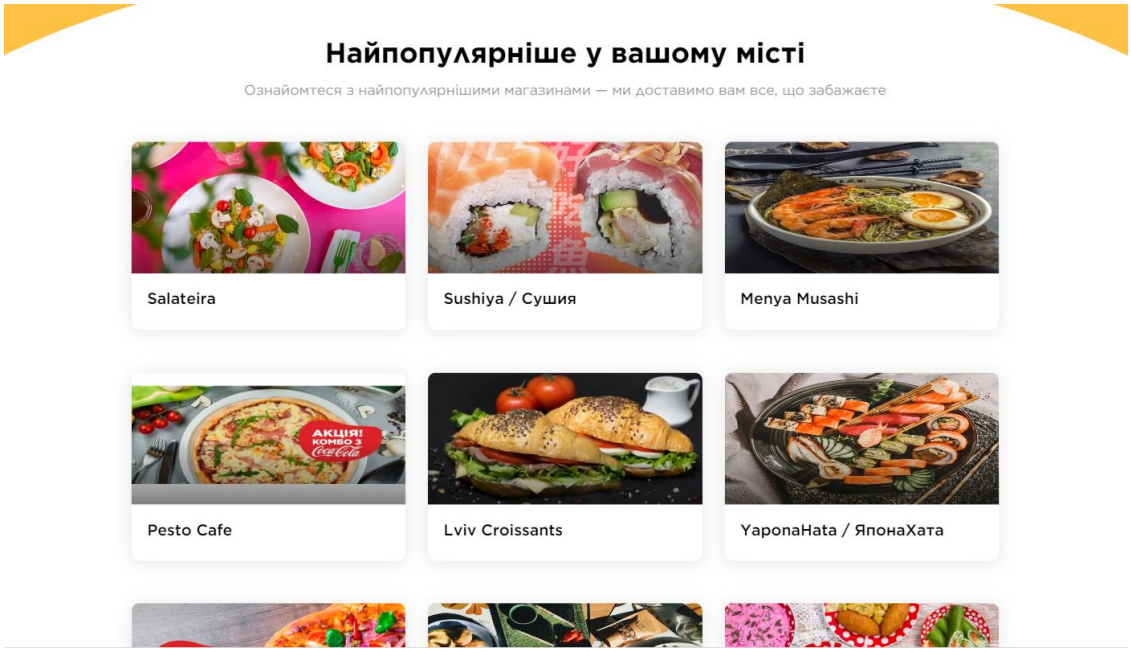


Рис. 1.2.2. Секція «Найпопулярніше» сервісу Glovo

Raketa(див.рис.1.2.3) зайшла на ринок після Glovo та UberEats, тому щоб залучити клієнтів вони заявили: доставка буде безкоштовною не тільки на період тестового режиму, а завжди. Проте я ставлю під сумнів таку заяву, адже

для достойного заробітку кур’єрів, на мою думку, має бути додаткова плата за кожне замовлення, яка буде йти з кишені замовника. У своєму веб-сервісі я буду реалізовувати замовлення враховуючи плату за доставку.

З недоліків: 1) відсутність фільтру по кухням: доводиться гортати всі заклади зі списку(див.рис.1.2.4); 2) не всі страви супроводжуються фотографією; 3) сторінку зі статусом замовлення потрібно оновлювати самостійно, в той час як у UberEats та Glovo це відбувається автоматично; 4) відсутня можливість оцінити роботу кур’єра та залишити відгук ресторану.

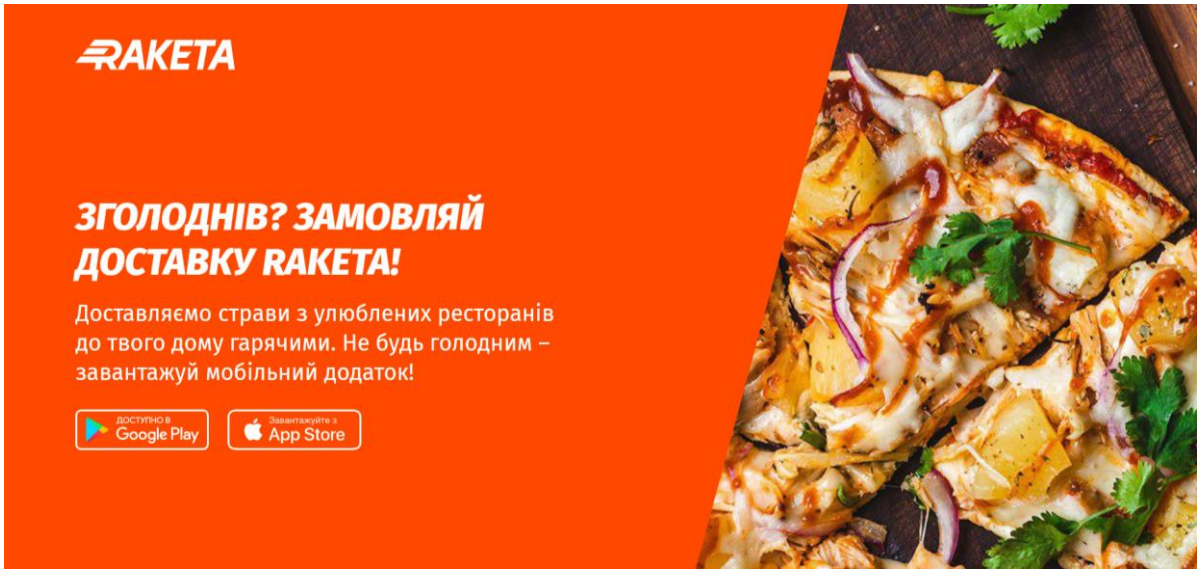


Рис. 1.2.3. Головна сторінка сервісу «Raketa»

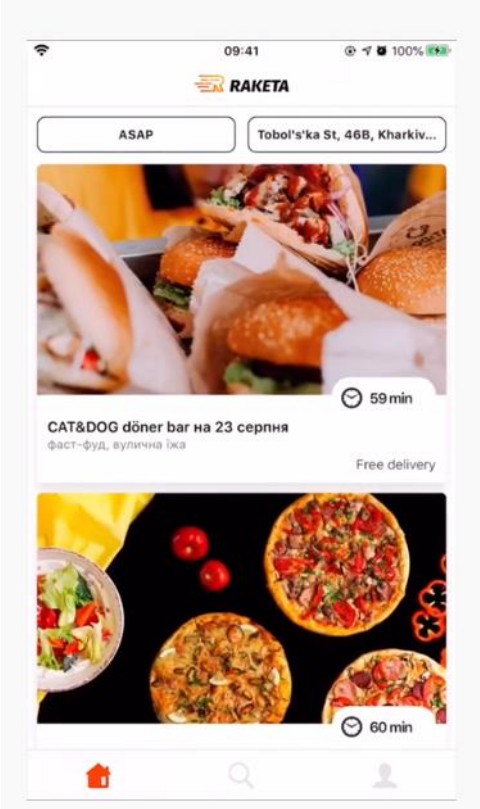


Рис. 1.2.4. Перегляд ресторанів у мобільному додатку Raketa

Також зазначу обмеження сервісу **UberEats**(див.рис.1.2.5, рис. 1.2.6): 1) заплатити можна тільки карткою; 2) можна замовити через сайт, тільки додаток.

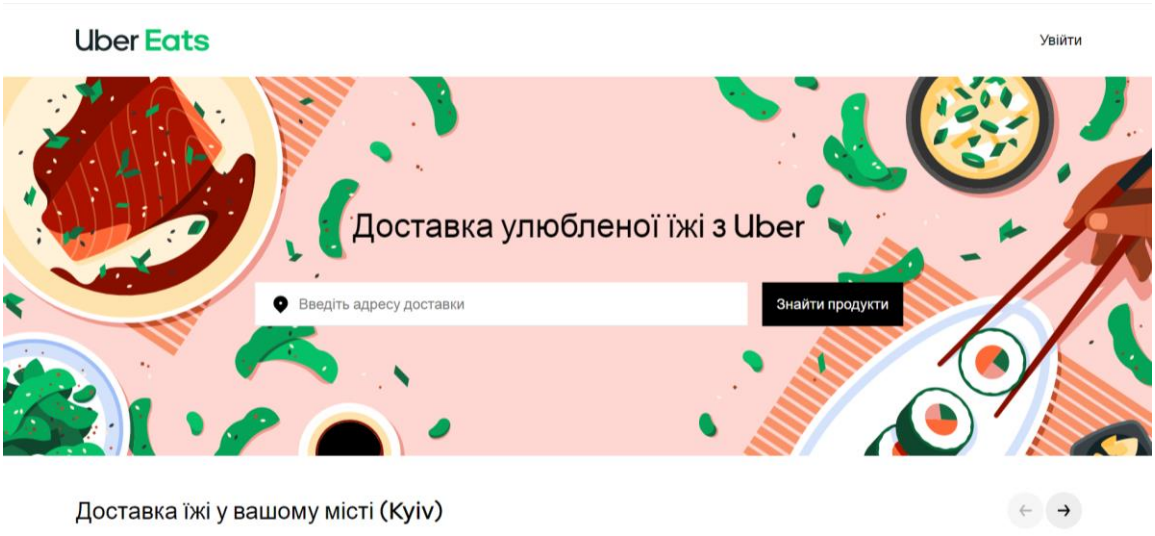


Рис. 1.2.5 Головна сторінка сервісу UberEats

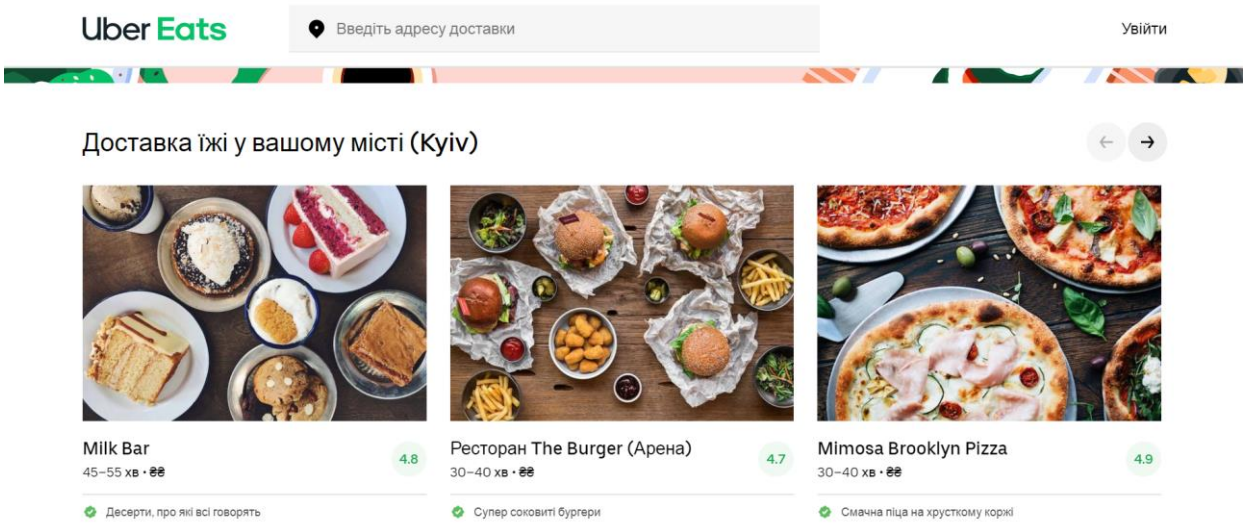


Рис. 1.2.6. Перегляд ресторанів сервісу UberEats

Отже, висновком є пропозиція таких шляхів усунення обмежень існуючих сервісів:

1. Для категорії продукт створити поле, в якому буде вказуватись, ціна за кг/упаковку/штуку.
2. Не створювати у сервісі секцію найпопулярніше, адже вона буде конфліктувати з пошуком магазинів у зоні, в якій знаходиться користувач.
3. Додати фільтри пошуку.
4. Поле «Фото страви» обов’язкове.
5. Кнопка переходу до перегляду власного замовлення у користувача має мати вигляд, асоційований із замовленням.

І найголовніша проблема, яка вирішується, це брак на ринку саме веб-додатків доставки. Адже велика кількість сервісів мають лише додаток, а не сайт, і користувач не має можливості замовити доставку без встановлення певної програми на свій пристрій.

1.3. Постановка завдання

1. Аналіз бізнес-логіки роботи, функціональних вимог та використаних засобів користувацьких інтерфейсів існуючих онлайн-сервісів доставки замовлень.
2. Дослідження та вибір найактуальніших технологій веб-розробки для створення сучасних веб-застосувань .
3. Розробка онлайн-сервісу доставки замовлень, що задовольняє таким функціональнім вимогам:
   - автентифікація та реєстрація користувачів;
   - перегляд ресторанів за категоріями;
   - перегляд сторінки ресторану: його товарів;
   - додавання у свій профіль контактної інформації;
   - оформлення замовлення: додавання товарів у корзину;
   - перегляд та зміна інформації про магазин, його страви, локації та кур’єрів (для адміністратора);
   - перегляд інформації по замовленням та клієнтам (для адміністратора).

РОЗДІЛ 2

2.1. Основні принципи розробки веб-застосувань

Проектування структури сайту ґрунтується на принципі клієнт-сервер та MVC шаблоном.

Клієнт, який є браузером, звертається до серверу, а сервер відповідає на його запит(див. рис. 2.1.1). Відповідно у ролі клієнта може виступати будь-який пристрій користувача: комп'ютер, мобільний телефон або інший гаджет. Таку взаємодію можна зобразити таким чином:

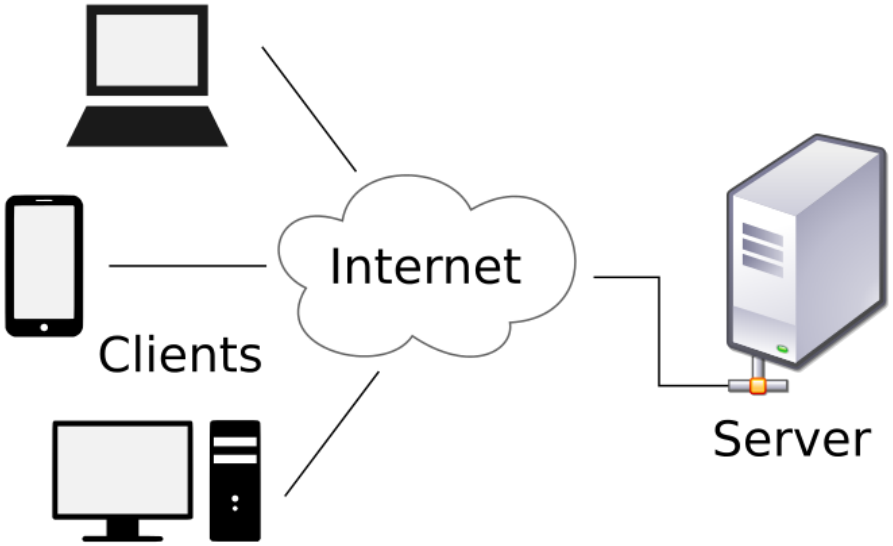


Рис 2.1.1. Схема клієнт-серверної архітектури

Шаблон проектування та розробки програмного забезпечення MVC [4] є абревіатурою Model View Controller. Контролери, представлення та моделі – це частини програми(див. рис.2.1.2), кожна з яких є групою файлів.

Контролер – це логіка додатку, зв'язок між Представленням та Моделлю. Він відповідає за url-адресу та дії, які можна виконати над об'єктом або ж які може виконати об'єкт. Це означає, що запит, який наявний у адресі, наприклад domain_name/shops/2, оброблює саме контролер. У файлах-

контролерах доволі часто можна зустріти умовні конструкції або ж світч-кейси.

Представлення – те, яким програму бачить користувач.

Модель відповідає за зв’язок із базою даних.

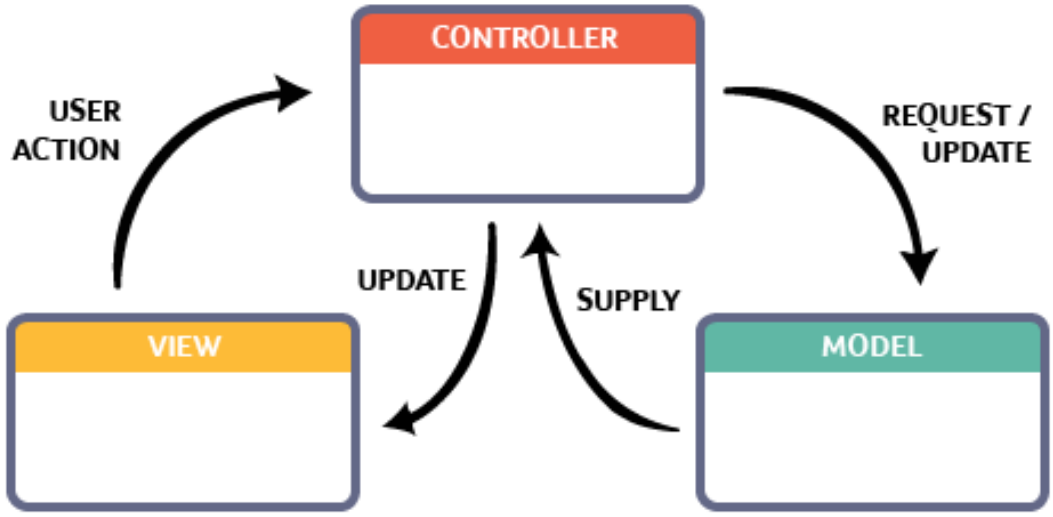


Рис 2.1.2. Принцип роботи MVC – паттерну

Будь-яке веб-застосування не буде існувати без 2 складових: браузера та користувача. Адже за допомогою першого другий взаємодіє із сервісом.

Цикл, за яким працює застосування, складається з таких кроків: 1) користувач вводить url-адресу сайту та браузер звертається до контролеру 2) контролер звертається до моделі 3) модель – до бази даних 4) з бази даних інформація передається у модель 5) з моделі у контролер 6) контролер передає інформацію у представлення 7) представлення відображаються у браузері за допомогою контролеру.

Фреймворк Django реалізовує подібну архітектуру – MVT (Model–View–Template).

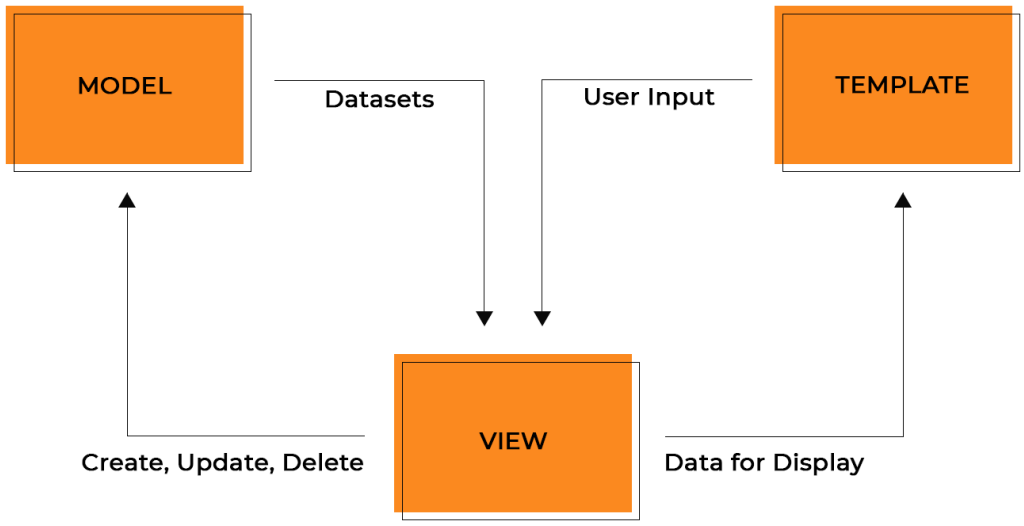


Рис. 2.1.3. MVT патерн

Контролер є посередником між користувачем та базою даних, адже працювати з базою напряму є поганою практикою. Наявність контролера розподіляє навантаження на систему та забезпечує вищий рівень безпеки даних. Дані у свою чергу представлені у вигляді об’єктів, які мають властивості (фактично атрибути сутностей бази даних) та функції. Вставка, вибірка, видалення та оновлення – операції, які можуть бути виконані за запитом контролера. Їх ще називають CRUD [5] – create-read-update-delete. Вони є базовими функціями управління даними.

Нижче наведена таблиця [6], яка зображає відповідність операцій запитам SQL та HTTP(див.рис.2.1.3).

Operation	SQL	HTTP
Create	INSERT	PUT / POST
Read (Retrieve)	SELECT	GET
Update (Modify)	UPDATE	PUT / POST / PATCH
Delete (Destroy)	DELETE	DELETE

Рис 2.1.3. Співвідношення операцій та запитів

2.2. Класифікація сайтів

Станом на січень 2020 року [7], кількість веб-сайтів у світі перевищувала 1.74 більйона. Їхній контент, мета та послуги залежать від компанії, країни та багатьох інших факторів. Проте все ж сайти можна об’єднати за спільними рисами. Для вивчення цього питання була опрацьована стаття контентного стратега Лексі Лу [8]. Щоб виявити належність веб-застосування для організації доставки замовлень якомусь із видів, розгляну кожен із них:

1. Homepage

Так званий сайт-домашня сторінка складається із однієї сторінки, навігація по якій відбувається зазвичай за посиланнями у верхньому меню. Це може бути сайт невеликого бренду. Можливість придбати товари, наприклад, може бути присутня, але для великої корпорації з широким асортиментом такий формат не буде доречним.

2. Magazine

Сайти журналів та видань мають характерно велику кількість текстів, фото та відео. Розробнику треба звертати увагу на коректне відображення тексту незалежно від його розміру, а також вирішити питання збереження медіа-контенту.

3. E-commerce

Веб-сервіси онлайн-покупок – сайти, якими користується кожен із нас. Rozetka, iHerb, Amazon та багато інших компаній належать цій категорії. Проте це непроста задача для реалізації. Адже після відображення товарів та додавання фільтрів за категоріями треба контролювати справність акаунту та підвантаження великої кількості інформації.

4. Blog

Деякі бренди та бізнеси ведуть свої сторінки у форматі блогу. Особливістю цього типу є те, що сторінки та їхній контент потрібно регулярно створювати та оновлювати, щоб блог не був застарілим.

5. Portfolio website

Сайти-портфоліо вимагають від розробників та дизайнерів максимальної креативності, адже кожен такий сайт є унікальним: письменник та актор будуть мати кардинально різні сторінки.

6. Landing page

Сторінка візитівка має за мету те, щоб користувач зацікавився продуктом та здійснив дію - call-to-action. Наприклад, заповнив форму, вказавши свою електронну пошту.

7. Social media website

Веб-сторінки соціальних мереж мають надзвичайно широкий функціонал: чати, канали, групи, заходи і т.д. Вони націлені на спілкування між людьми та постійно вдосконалюють можливості користувача. Більше того, розробники застосовують технології AI.

8. Contact page

Контактні сторінки надають можливість переглянути список за певним критерієм. Наприклад, всі ресторани міста Києва. Контактними можна назвати сайти пошуку роботи.

З наведеного огляду можна зробити висновок про те, що створюване в даній роботі веб-застосування належить до категорії e-commerce, проте має спільні риси (одну сторінку, по якій відбувається навігація) із сайтом накшталдт домашньої сторінки.

РОЗДІЛ 3

3.1 Аналіз технічного завдання

У сервісу наявні наступні групи користувачів: адміністратор, кур’єр та замовник. Адміністратор має доступ до статистичних даних сервісу, має можливість вносити та змінювати інформацію про кур’єрів, магазини та їхні товари. Кур’єр має доступ до локацій замовника та магазину, звідки замовник чекає товар, та до детальної інформації по замовленню. Замовник має можливість створити свій акаунт, створити замовлення, має доступ до інформації про магазини, товари, ціни, кур’єра, який буде доставляти замовлення. У сервісі реалізовані інтерфейси адміністратора та замовника.

Технічні вимоги користувачів до функціональності системи:

Адміністратор:

1. Перегляд та зміна інформації про магазин, його страви, локації та кур’єрів.
2. Перегляд інформації по замовленням та клієнтам.

Замовник:

1. Реєстрація та автентифікація.
2. Перегляд та зміна інформації акаунту: імені, адреси, телефону, номеру платіжної картки.
3. Перегляд магазинів за категоріями.
4. Перегляд товарів магазину.
5. Створення замовлення(1 замовлення тільки у 1 магазину): обирання товарів, їх кількості, зазначення приміток до замовлення, способу оплати, адреси.
6. Оформлення замовлення, а саме натискання кнопки, після якого замовлення стане видно кур’єрам, і замовник вже не зможе внести зміни до замовлення.
7. Перегляд інформації про замовлення, яке зараз виконується.

Специфікація вимог до даних:

- **Магазин**

  Магазин містить страви. Про магазин зберігається назва, опис, категорія та локація(1 або більше). Інформацію про магазин додає, змінює та видаляє адміністратор. Клієнт та адміністратор можуть переглядати повну інформацію про всі магазини. Кур’єр переглядає тільки назву та локацію магазину, звідки здійснює замовлення.

- **Товар**

  Про товар зберігається унікальний номер, назва, категорія, фото, опис, вага, ціна, одиниця виміру. Товар належить магазину та не може існувати окремо. Якщо товар – це страва, то вказується вага. Якщо товар – це продукт, як наприклад мандарин, то вказується ціна та одиниця виміру: штука/кілограм абощо. Адміністратор переглядає інформацію про всі товари, кур’єр – лише про ті, які доставляє. Клієнт може переглядати інформацію про всі товари всіх магазинів, та ті, які замовив.

- **Кур’єр**

  Про нього зберігається унікальний номер, ім’я, фото, телефон та статус активності. Клієнт, чиє замовлення здійснює кур’єр, має доступ до його ім’я, телефону та фото. Адміністратор має доступ до повної інформації про всіх кур’єрів. Кур’єр може здійснювати 1 замовлення за раз, тобто не декілька замовлень одночасно.

- **Замовлення**

  Створюється одним клієнтом, здійснюється одним кур’єром та з одного закладу. Містить будь-яку кількість різних страв. Кур’єр бере замовлення, тим самим додаючи у замовлення свій унікальний номер, та змінює статус замовлення під час його виконання. Кур’єр має можливість переглянути локації магазину та клієнта до того як візьме

замовлення, але лише тоді, коли локація магазину знаходиться в радіусі «поблизу». Кур’єр має можливість переглянути деталі замовлення та контакт клієнта якщо він взяв замовлення. Замовлення містить унікальний номер, примітки, чи оплачено, статус виконання, дату та час оформлення замовлення та спосіб оплати. Адміністратор може переглядати дані про всі замовлення. Клієнт може переглядати замовлення, але змінити його не може.

- **Клієнт**

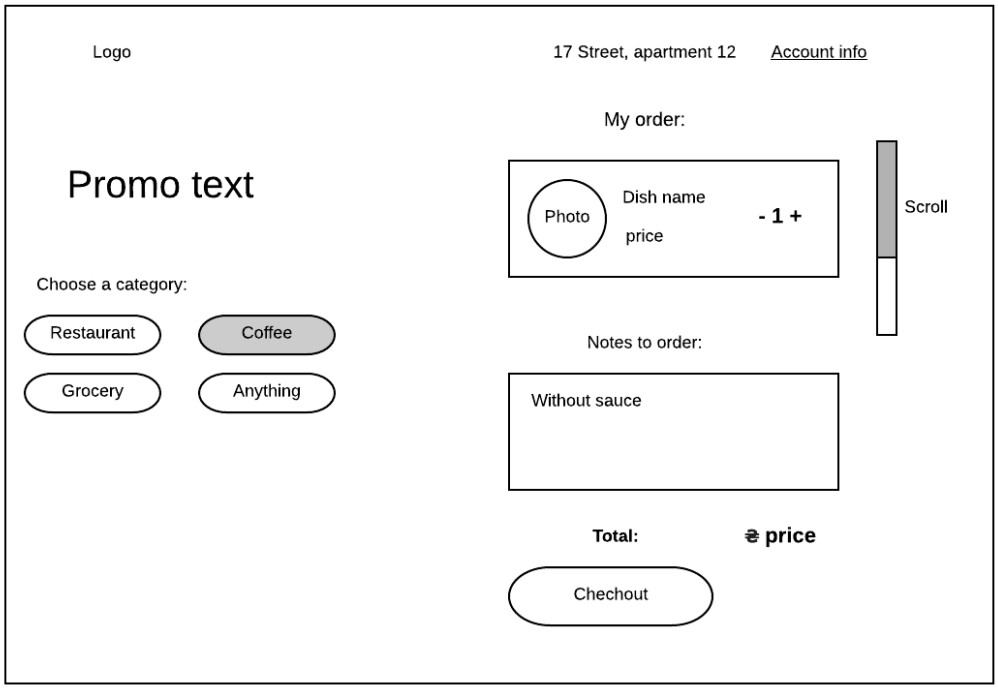
  Клієнт має акаунт, який містить ім’я, телефон, адресу, номер платіжної картки та адресу електронної пошти. Доступ до контактних даних має кур’єр, який здійснює замовлення клієнта. Одночасно може виконуватись 1 та більше замовлень одного клієнта. Адміністратор має доступ до повної інформації про всіх клієнтів.

Кожен користувач має можливість увійти за вийти із сервісу. Клієнт може зареєструватись самостійно через інтерфейс сервісу. Адміністратори реєструють нових кур’єрів та адміністраторів.

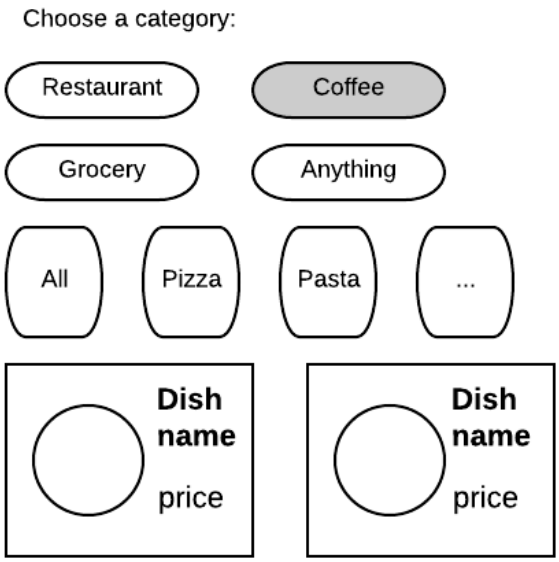
Інтерфейси:

Перед початком розробки та верстки дизайну, для зрозумілішого відображення концепції додатку та способів взаємодії з ним, було побудовано схеми-інтерфейси. Вони демонструють представлення сайту у браузері. Та на їх основі розроблюється веб-дизайн.

Головна сторінка:

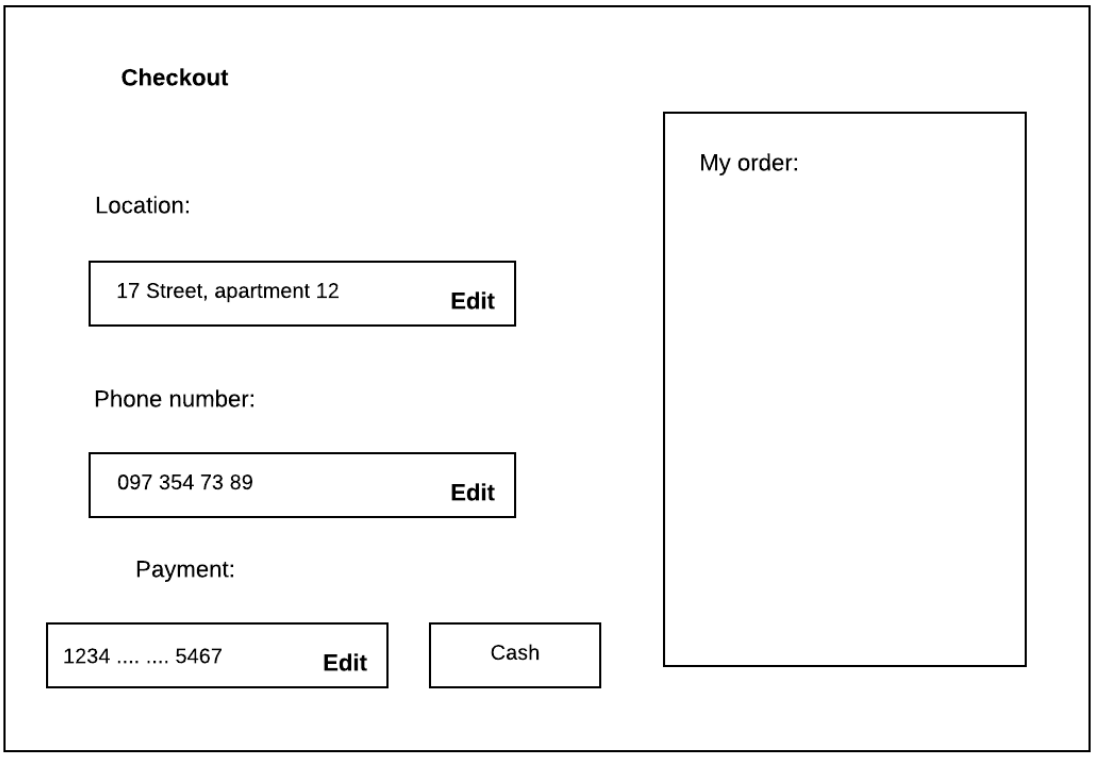


При кліку на категорії Coffee, Restaurant та Grocery:

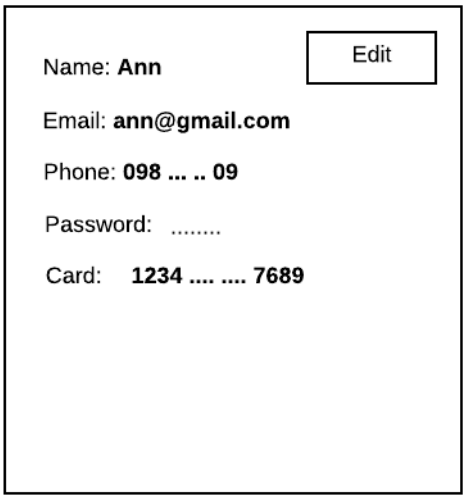


При кліку на Anything:

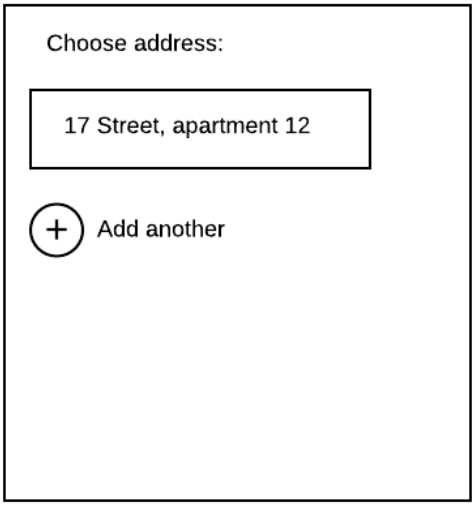
При кліку на Checkout:



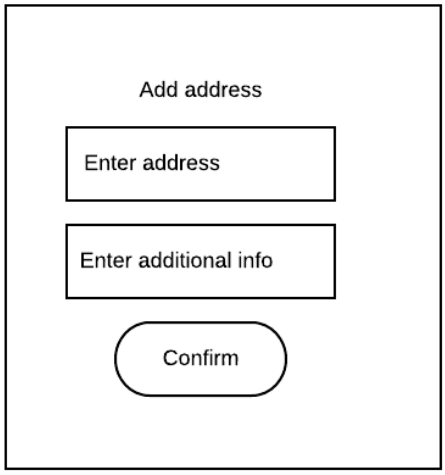
При кліку на Account info:



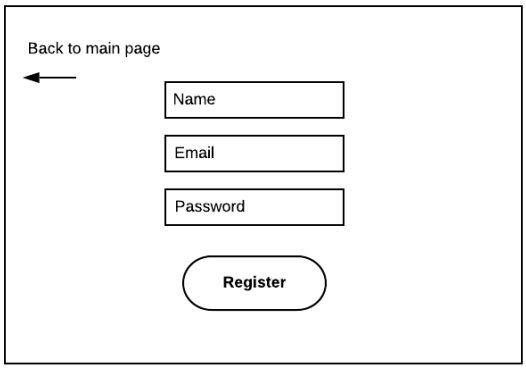
При кліку на локацію:



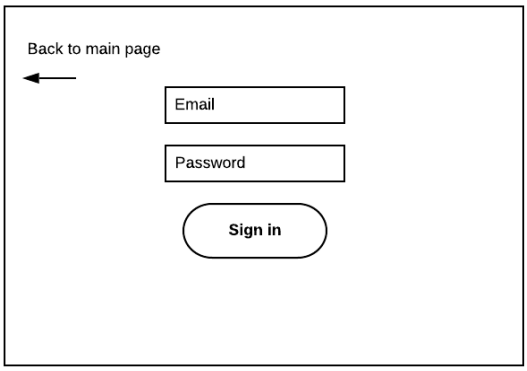
Додати ще одну адресу:



Реєстрація:



Вхід:



3.2 Обґрунтування алгоритму й структури програми

Алгоритм роботи програми зображений у вигляді блок-схеми.

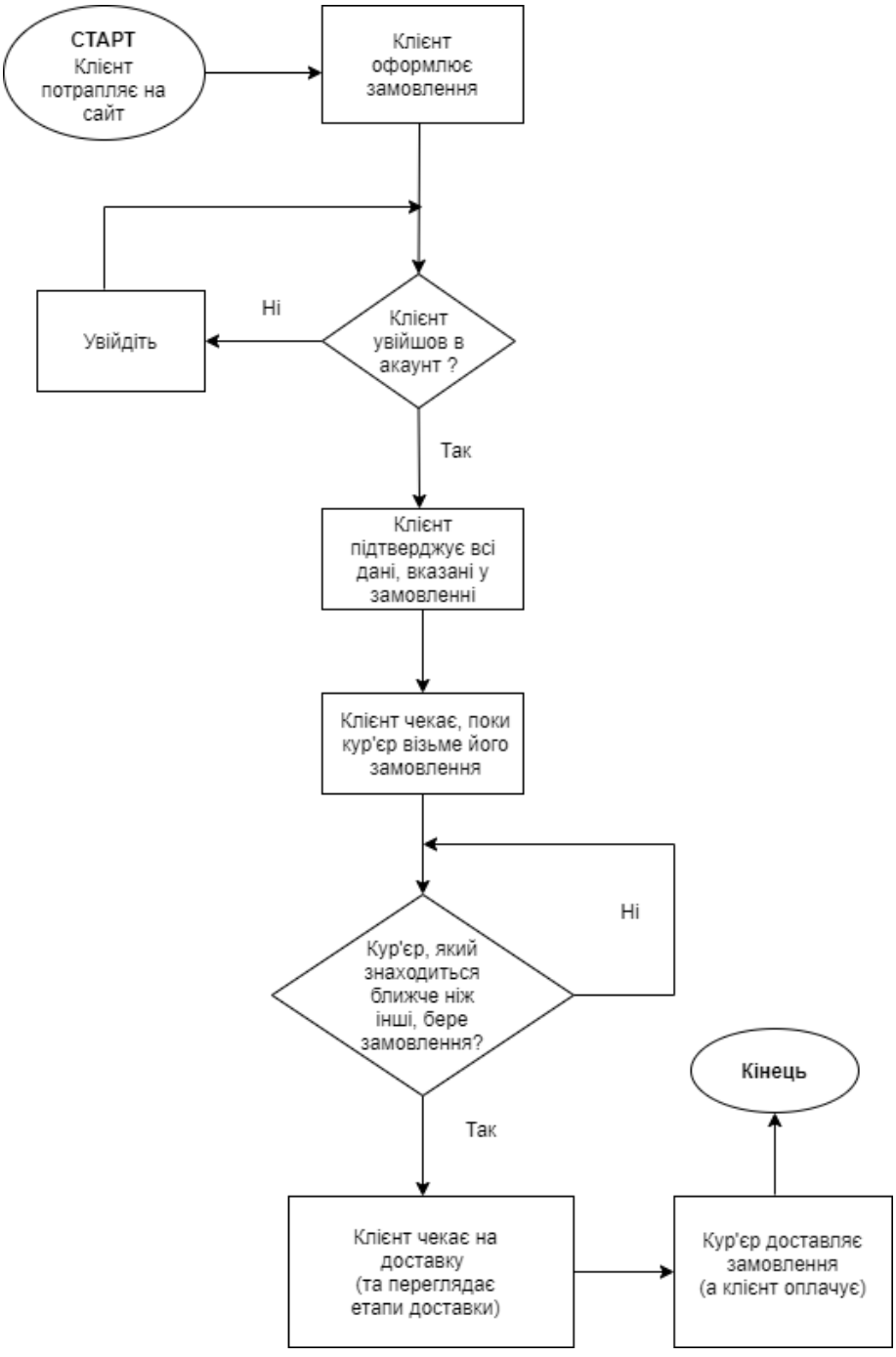


Рис. 3.2.1 Діаграма алгоритму роботи програми

Проаналізувавши побудовану схему (див.рис.3.2.1), можна виділити основні модулі програми: реєстрація та автентифікація, оформлення,

редагування деталей замовлення, модуль очікування на початок доставки, очікування доставки, завершення процесу доставки.

3.3 ***Обгрунтування вибору засобів розробки***

Для розробки свого застосування я обрала мову Python [9]. Вона є однією із найпопулярніших мов програмування нашого часу і нею користуються як розробники, які працюють у сфері машинного навчання, так і веб-девелопери. Чому ж мова користується таким опитом?

- Вбудовані бібліотеки.

Стандартна бібліотека включає в себе інтернет-протоколи, інструменти веб-служб та інтерфейси операційної системи. Рішення задач, які часто виникають під час програмування, вже є у бібліотеці, що значно скорочує довжину коду. А до цього і прагне кожен розробник.

- Доступність навчання та зручність роботи.

Читабельність коду, простий зрозумілий синтаксис та настанови щодо стилю коду PEP 8 сприяють тому, що розробники обирають цю мову.

- Структури даних.

Вбудовані список та словник використовуються для побудови структур даних, які мають швидкий runtime – час виконання. Більше того, Python надає можливість динамічного введення даних на високому рівні.

- Швидкість та продуктивність.

Реалізовує об’єктно-орієнтовану парадигму, надає можливості управління процесами, інтеграції, обробки тексту. Також має власний unit-testing фреймворк.

Разом із мовою Python я використовую фреймворк Django [10]. В багатьох веб-застосуваннях є потреба у контролі складних зв’язків між даними, можливості додавати користувачів, автентифікувати, надавати користувачам доступ до оновлення інформації і т.д. Фреймворк Django надає

саме такі можливості, так звані out-of-the-box. Django побудований на ідеї python-пакетів – колекції python-файлів, які згуртовані для певної мети, вирішення конкретної задачі. Фреймворк дозволяє створити веб-застосунок, який інтегрує пакети та дозволяє розробнику зручно розбити свій проект на модулі.

Django забезпечує:

- Скорочення шляху від створення концепції програми до завершення її розробки. Завдяки розбиттю на файли, фреймворк надає можливості для швидкої та ефективної розробки. Кожен файл виконує свою окрему задачу.
- Десятки додаткових бібліотек, які можна використовувати для вирішення загальних завдань веб-розробки. Django піклується про автентифікацію користувачів, адміністрування вмісту та інше.
- Надійний захист даних. Він допомагає розробникам уникнути багатьох поширених помилок, пов’язаних із безпекою веб-застосунків, таких як SQL-ін’єкції, clickjacking-атаки та крос-сайтові запитові підробки запитів. Система автентифікації користувачів надає безпечний спосіб управління обліковими записами та паролями користувачів.
- Можливість швидко та гнучко масштабуватись, щоб задовольнити вимоги великого трафіку.

Фронт-енд розробка виконувалась за допомогою HTML, CSS та JavaScript (ECMAScript 6). Мова розмітки дозволила розбити сторінку на контекстні блоки, мова стилю сторінок – надати їм якісний UI/UX, а JavaScript – динамічності сторінкам. Наприклад, виконання певних дій під час кліку на кнопку.

Для збереження та контролю над базою даних було використано об'єктно-реляційну систему керування базами даних PostgreSQL [11]. Вона

використовує та розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають та масштабують найскладніші навантаження даних.

Для реалізації проекту – безпосередньо написання коду – було обрано середовище розробки PyCharm Professional [12] від компанії JetBrains. Це IDE для професійної розробки на мові Python. Механізм аналізу коду забезпечує точне автодоповнення, пошук помилок, швидкі виправлення та зручну навігацію по коду. Також контролює якість коду за допомогою перевірок відповідності вимогам PEP8 та розумному рефакторингу.

Вбудований термінал дозволяє вносити зміни у різні модулі проекту та встановлювати бібліотеки, а інтеграція з популярними системами контролю версій – розроблювати програму із документацією усіх змін та пояснень до них(commits). Таким чином, зберігати проект та всі його версії на GitHub стало для мене задачею, максимально комфортною для здійснення.

Однією з найзручніших для мене функцій є можливість додати у вікно IDE перегляд бази даних, яка використовується у проекті. За допомогою цього інструменту можна не тільки переглядати дані, які зберігаються у базі, а й додавати нові таблиці та їхні атрибути, змінювати їх та видаляти. Для роботи над проектами із базою даних, яка має велику кількість зв’язків, це економить час та зусилля та дозволяє уникнути помилок, адже PyCharm надає лаконічний та зрозумілий інтерфейс.

Окрім Python, PyCharm підтримує JavaScript, SQL, HTML/CSS, мови шаблонів та інші технології. Це дозволило мені працювати над версткою, динамічністю сайту та базою даних в одному середовищі.

3.4 Опис розробки програм

Застосування побудоване на принципі клієнт-серверної архітектури, яка визначає 3 типи компонентів: сервер, клієнт та мережа. Сервери надають

інформацію та послуги, клієнти використовують сервіси, які надаються серверами. А мережа забезпечує взаємодію між клієнтами та серверами.

З’єднання здійснюється за HTTP-протоколом: клієнт(браузер) надсилає http-запит на сервер, а сервер надсилає відповідь клієнту. GET-запит використовується для того, щоб отримати дані з сервера. POST – щоб надіслати дані серверу, наприклад, додати чи оновити дані. Запит у програмному коді виглядає наступним чином:

```
first_name = request.POST['first_name']
```

Програма відповідає схемі дизайну MVC, яка розділяє веб-інтерфейс та бізнес-логіку. Такий підхід допомагає розробникам спростити та пришвидшити розробку великих веб-додатків. Проект складається із пайтон-файлів, кожен з яких виконує певну задачу:

views.py – є свого роду контролером, який поєднує відображення інформації клієнту(html-сторінки) та моделі таблиць бази даних. Складається з функцій, кожна з яких відповідає за певний запит від клієнта сервісу. Відповідно у цих функціях оброблюються помилки, які можуть виникнути у наслідок дій користувача.

```
from django.shortcuts import render
from .models import Shop
from django.http import Http404


def index(request):
    context = {
        "shops": Shop.objects.all()
    }
    return render(request, "client/index.html", context)


def shop(request, shop_id):
    try:
        shop = Shop.objects.get(pk=shop_id)
    except Shop.DoesNotExist:
        raise Http404("Shop does no exist")
    context = {
        "shop": shop
    }
    return render(request, "client/shop.html", context)
```

urls.py – вказує яка функція з views.py відповідає за відображення сторінки за певним адресним рядком.

```
from django.urls import path
from . import views

urlpatterns = [
    path("", views.first),
    path("register", views.register),
    path("login", views.login),
    path("logout", views.logout),
    path("add_user_info", views.add_user_info),
    path("add_user_info_submit", views.add_user_info_submit),
    path("shops", views.index),
    path("shops/<int:shop_id>", views.shop)
]
```

models.py – зберігає моделі, які є представленням таблиць бази даних. Кожна модель – це клас, його поля – атрибути таблиці.

```
class Dish(models.Model):
    name = models.CharField(max_length=64)
    category = models.CharField(max_length=64)
    description = models.CharField(max_length=64)
    weight = models.IntegerField()
    shop = models.ForeignKey(Shop, on_delete=models.CASCADE, related_name="dishes",
default=None)
```

Також присутня папка із html-сторінками.

Відображення об’єктів програми на ці сторінки відбувається шляхом мови шаблонів django, а перевірка на автентифікацію за допомогою функції моделі User.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>First page</title>
</head>
<body>
{% if user.is_authenticated %}
    <h2>Hello, {{ user.first_name }}</h2>
    <button><a href="logout">Log out</a></button>
    <button><a href="add_user_info">Add info</a></button>

{% else %}
    <button><a href="register">Register</a></button>
    <button><a href="login">Login</a></button>
```

```
{% endif %}
</body>
</html>
```

Всі зміни, внесені у базу даних зберігаються у пайтон-файлах міграцій. Додавання нового поля до моделі виглядає таким чином:

```
from django.db import migrations, models

class Migration(migrations.Migration):

    dependencies = [
        ('client', '0004_auto_20200307_1644'),
    ]

    operations = [
        migrations.AddField(
            model_name='shop',
            name='image',

field=models.ImageField(default='https://res.cloudinary.com/glovoapp/w_680,h_240,c_fit,f_auto,q_auto/Products/phfj7p3w7dhjpxzovqk6', upload_to=''),
        ),
    ]
```

Не менш важливим є файл **settings.py**, який містить інформацію про всі бібліотеки, які використовуються, базу даних, ключі безпеки та валідатори паролів. Загалом, всі найважливіші налаштування проекту. Вони зберігаються у форматі ключ-значення. Наприклад, інформація про базу даних має такий вигляд:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'deliverydb',
        'USER': 'postgres',
        'PASSWORD': 'milena',
        'HOST': 'localhost',
        'PORT': '5432'
    }
}
```

Мова шаблонів django, яка використовується у проекті, надає можливості виводити інформацію, яка передається з контролеру представленню, циклом, використовувати умовні оператори та включати html-сторінки або ж наслідуватись від них. Безперечно, це дозволяє розробнику

писати лаконічний код, який не буде містити дублікатів, та буде читабельним. Наведу кілька прикладів:

Наслідування: `{% extends "base.html" %}`

Умовна конструкція:
```
{% if shop.category == "Restaurant" %}
<div>{{ shop.name }}</div>
{% endif %}
```

Для адміністрування сайту – перегляду та зміни інформації про кожен об’єкт програми – використовується Django admin. Сторінка адміністратора знаходиться за адресою /admin.

Створення об'єктів і розробка головної програми

Створення об’єктів відбувається у файлі views.py. Воно відбувається шляхом взаємодії із моделлю об’єкта за допомогою методів бібліотеки django.db – models.

Наприклад, масив об’єктів магазинів створюється шляхом створенням контексту, який має значення Shop.objects.all().

```
"shops": Shop.objects.all()
shop = Shop.objects.get(pk=shop_id)
```

Створення одного із найголовніших об’єктів програми – користувача – відбувається за допомогою модулю auth фреймворку Django. Разом з об’єктом, який зберігає такі дані як ім’я, пароль і пошту, створюється сесія користувача.

Пароль зберігається у форматі `<algorithm>$<iterations>$<salt>$<hash>.`

Для нього використовується алгоритм PBKDF2 із SHA256 хешуванням – механізм, рекомендований NIST – Національним інститутом стандартів і технологій.

Дані про користувача зберігаються у двох зв’язаних між собою моделях. Модель Client створено мною та містить дані, які характерні конкретно для

мого проекту, як наприклад номер телефону користувача. Auth_User – модель фреймворку Django, яка надає можливість безпечно зберігати паролі користувачів, надавати права доступу та створювати сесії.

Обробка помилок, які виникають внаслідок відсутності об’єкта, наприклад, за певним індексом, відбувається за допомогою класу DoesNotExist бібліотеки models та Http404 бібліотеки http.

3.5 *Опис файлів даних та інтерфейсу програми*

Головна сторінка сайту, на яку потрапляє користувач, відображає всі можливі категорії товарів для замовлення:

```
<div class="shop-categories-nav">
    {% for category in shop_categories %}
        <button class="shop-category-button"> {{ category }}</button>
    {% endfor %}
</div>
```

А також навігацію на вікно авторизації та реєстрації.

```
<button><a href="register">Register</a></button>
<button><a href="login">Login</a></button>
```

Авторизація користувача відбувається за допомогою POST-запитів, звертання до моделі автентифікації auth. За допомогою функцій Django render та redirect відбувається перехід на сторінку залежно від валідності даних, введених користувачем.

```
def login(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']

        user = auth.authenticate(username=username, password=password)

        if user is not None:
            auth.login(request, user)
            return redirect("/")
        else:
            messages.info(request, "Invalid credentials")
            return redirect('/login')
    else:
        return render(request, 'client/login.html')
```

Процес регістрації включає в себе створення об’єкта User відповідної моделі з перевіркою чи існує вже в базі даних такий користувач. Перевірка відбувається за допомогою функції filter, а створення – create_user.

```
def register(request):
    if request.method == 'POST':
        first_name = request.POST['first_name']
        last_name = request.POST['last_name']
        username = request.POST['username']
        password1 = request.POST['password1']
        password2 = request.POST['password2']
        email = request.POST['email']
        if password1 == password2:
            if User.objects.filter(username=username).exists():
                messages.info(request, "Username taken")
                return redirect('/register')
            elif User.objects.filter(email=email).exists():
                messages.info(request, "Email taken")
                return redirect('/register')
            else:
                user = User.objects.create_user(username=username,
password=password1, email=email,
                                                first_name=first_name,
last_name=last_name)
                user.save()
                print("User created")
                return redirect('/login')
        else:
            messages.info(request, "Passwords not matching")
            return redirect('/register')
    else:
        return render(request, "client/register.html")
```

Повідомлення про неправильно вказаний пароль, зарезервовану електронну пошту користувача та інші сповіщення відображаються за допомогою messages.info.

Після успішної реєстрації або авторизації користувач потрапляє на головну сторінку, на якій навігація замінюється на кнопку акаунту та панель із замовленням.

Користувач може додати контактну інформацію в акаунт. Для кожного такого поля створена форма, яка додає його у базу даних POST-запитом.

Файл forms.py:

```
class CardForm(forms.Form):
    card = forms.CharField(label='Card', max_length=16)
```

Файл sidebar.html:

```
{% if client.card == "" %}
    <div class="account-input account-card">
        <form action="/" method="post">
            {% csrf_token %}
            {{ card_form }}
            <button id="btnAdd2" class="account-add" type="submit">Add</button>
        </form>
    </div>
{% else %}
    <div class="account-component">
        <label for="card">Card</label>
        <div id="card" class="account-value">
            {{ client.card }}
        </div>
    </div>
{% endif %}
```

При кліку на акаунт, з’являється pop-up вікно, у якому можна переглянути та змінити дані користувача. На панель замовлення можна додати бажані товари за допомогою натискання на них.

Кнопка Order завершує оформлення замовлення.

Дані про користувача зберігаються у файлах даних таким чином:

```
name='Client',
fields=[
    ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
    ('phone', models.CharField(max_length=13)),
    ('address', models.CharField(max_length=13)),
    ('password', models.CharField(max_length=64)),
    ('card', models.CharField(max_length=16)),
    ('email', models.EmailField(max_length=254)),
],
```

Про страву:

```
name='Dish',
fields=[
    ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
    ('name', models.CharField(max_length=64)),
    ('category', models.CharField(max_length=64)),
    ('description', models.CharField(max_length=64)),
    ('weight', models.IntegerField()),
],
```

Про кур’єра:

```
name='Courier',
fields=[
    ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
    ('phone', models.CharField(max_length=13)),
    ('name', models.CharField(max_length=64)),
],
```

Магазин:

```
name='Shop',
fields=[
    ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
    ('name', models.CharField(max_length=64)),
    ('description', models.CharField(max_length=64)),
    ('category', models.CharField(max_length=64)),
],
```

Замовлення:

```
name='Order',
fields=[
    ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
    ('notes', models.CharField(max_length=64)),
    ('paid', models.BooleanField()),
    ('delivered', models.BooleanField()),
    ('date', models.DateTimeField()),
    ('payment', models.BooleanField()),
    ('client', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE,
related_name='orders', to='client.Client')),
    ('courier', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE,
related_name='orders', to='client.Courier')),
],
```

Локацію магазину:

```
name='Location',
fields=[
    ('id', models.AutoField(auto_created=True, primary_key=True, serialize=False,
verbose_name='ID')),
    ('address', models.CharField(max_length=64)),
    ('shop', models.ForeignKey(on_delete=django.db.models.deletion.CASCADE,
related_name='locations', to='client.Shop')),
],
```

Файли даних зберігаються у класах міграцій: для кожної зміни, внесеної в базу даних, створюється клас Migration. Таблиця бази даних є моделлю, а для її полів зазначено їхні типи, максимальну довжину рядка(якщо поле типу char), чи є поле первинним ключем та інші деталі. Також у файлах даних зберігаються проміжні таблиці для зв’язку «багато-до-багатьох», такі як DishOrder.

База даних(див. додаток А) має такі таблиці: Замовлення, Клієнт, Кур’єр, Страва, Магазин. Також для збереження кількох локацій магазину додана

таблиця Локація. Оскільки зв'язок таблиць Замовлення та Страва – «багато-до-багатьох» - додана таблиця Замовлення-Страва.

Таблиці, згенеровані фреймворком Django:

django_migrations – зберігає інформацію про всі зміни, здійснені над базою даних: в якому модулі програми створена міграція, її назва, дата та час.

django_session – акумулює сесії юзерів сайту.

auth_user – зберігає інформацію про всі входи (автентифікації) у систему: пароль, дату та час останнього входу, ім'я користувача.

auth_user_groups – таблиця, що з'єднує користувачів та їх групи.

auth_group - визначає групи користувачів системи.

auth_group_permissions – визначає які права доступу до бази даних має кожна група користувачів.

django_admin_log – зберігає інформацію по зміни до бази даних, здійснені адміністратором.

django_content_type – таблиці бази даних та модулі програми, які їх контролюють.

auth_user_user_premissions – які права доступу має користувач із таблиці User (має зв'язок із таблицею Клієнт).

auth_permission – список всій дій, які можна здійснити над базою даних (наприклад, додати чи видалити страву і т.д.).

Перетворення таблиць у об'єкти, готові до взаємодії у програмному коді, відбувається у файлі models.py. Кожній таблиці відповідає модель – клас із полями. Також кожному такому класу можна перевизначити функцію виводу, таким чином зрегулювати як відобразиться об'єкт класу.

Наприклад, модель таблиці страви має вигляд:

```
class Dish(models.Model):
    name = models.CharField(max_length=64)
    category = models.CharField(max_length=64)
    description = models.CharField(max_length=64)
    weight = models.IntegerField()
    shop = models.ForeignKey(Shop, on_delete=models.CASCADE, related_name="dishes",
default=None)
```

```
    def __str__(self):
        return f"{self.id} : name {self.name}, category {self.category}, desc
{self.description}, " \
               f"weight {self.weight}"
```

3.7. Тестування програми і результати її виконання

Вхідними даними програми є запити користувача. Вихідними – результати, які повертаються з контролер-частини програми та виводяться на сторінку.

Перша сторінка, на яку потрапляє користувач, містить категорії страв, які є свого роду фільтрами. Щоб навіть не авторизований користувач міг переглянути асортимент. У правій частині екрану закріплений sidebar – він містить всі деталі по замовленню. Якщо користувач не авторизований, до замовлення не можна додати товари.

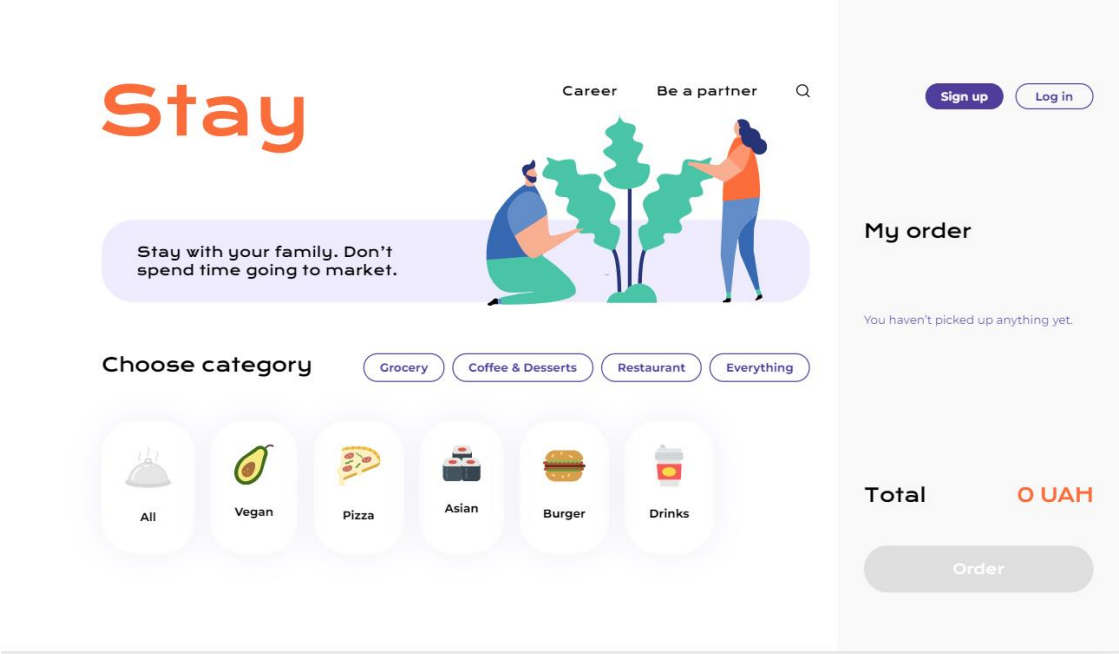


Рис.3.7.1 Головна сторінка сервісу

При кліку на категорію страви з’являються магазини, в яких присутні страви цієї категорії.

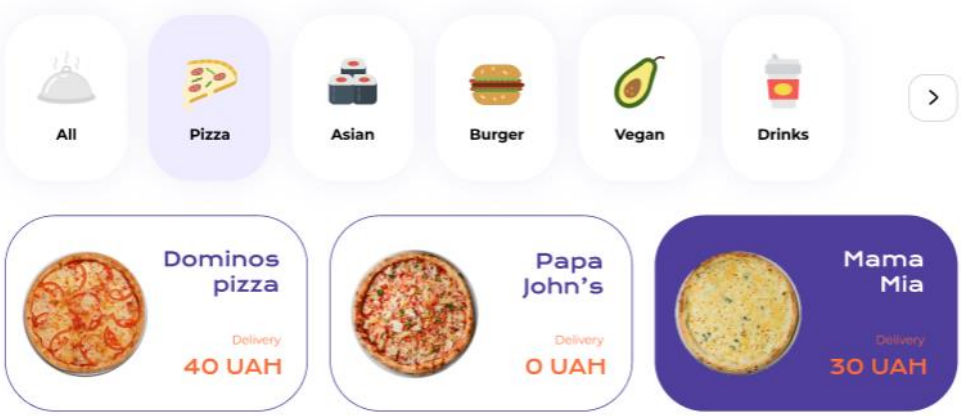


Рис. 3.7.2. Перегляд ресторанів за категоріями

Вікно реєстрації виглядає таким чином:

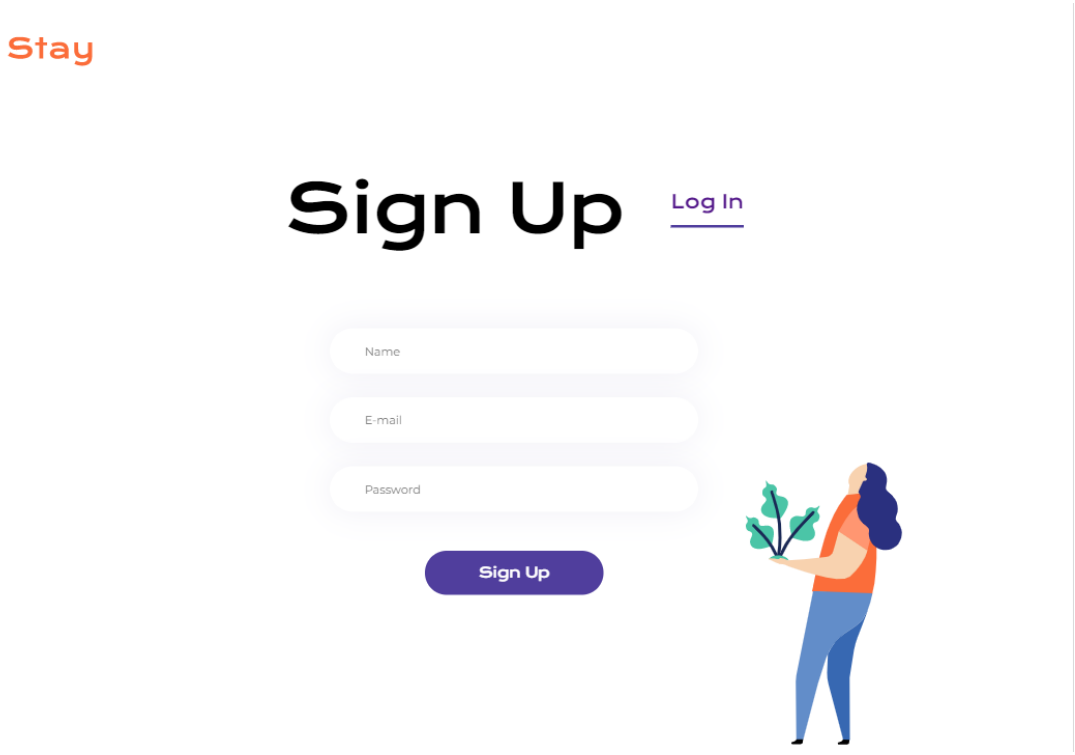


Рис.3.7.3. Сторінка реєстрації користувача

Входу – так:

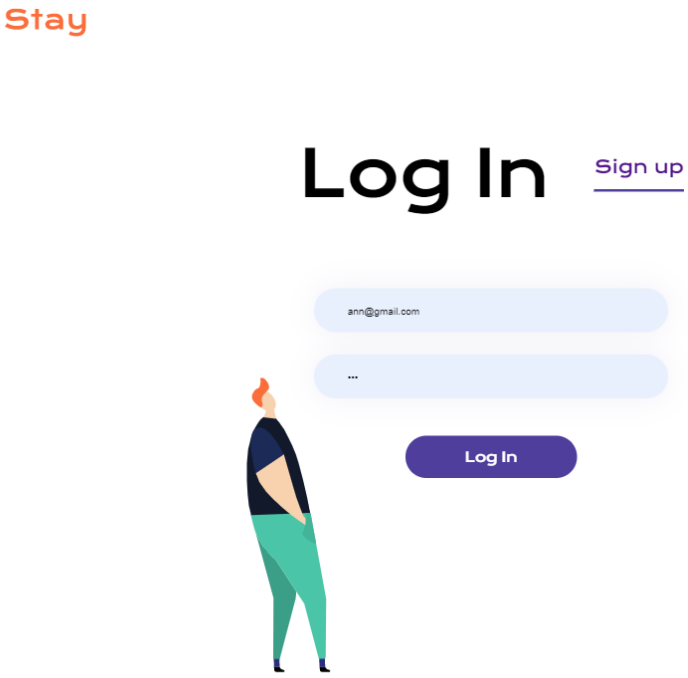


Рис.3.7.4. Сторінка автентифікації користувача

Якщо користувач здійснив авторизацію, він має можливість додати товари у замовлення.

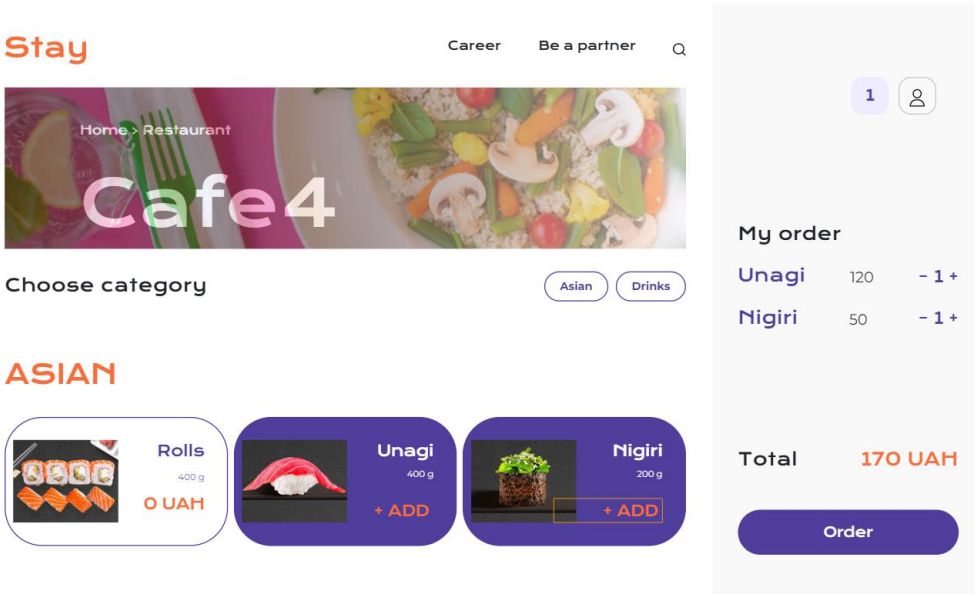


Рис.3.7.5. Сторінка ресторану

Контент sidebar-у змінюється при кліку на іконки у його верхньому меню.

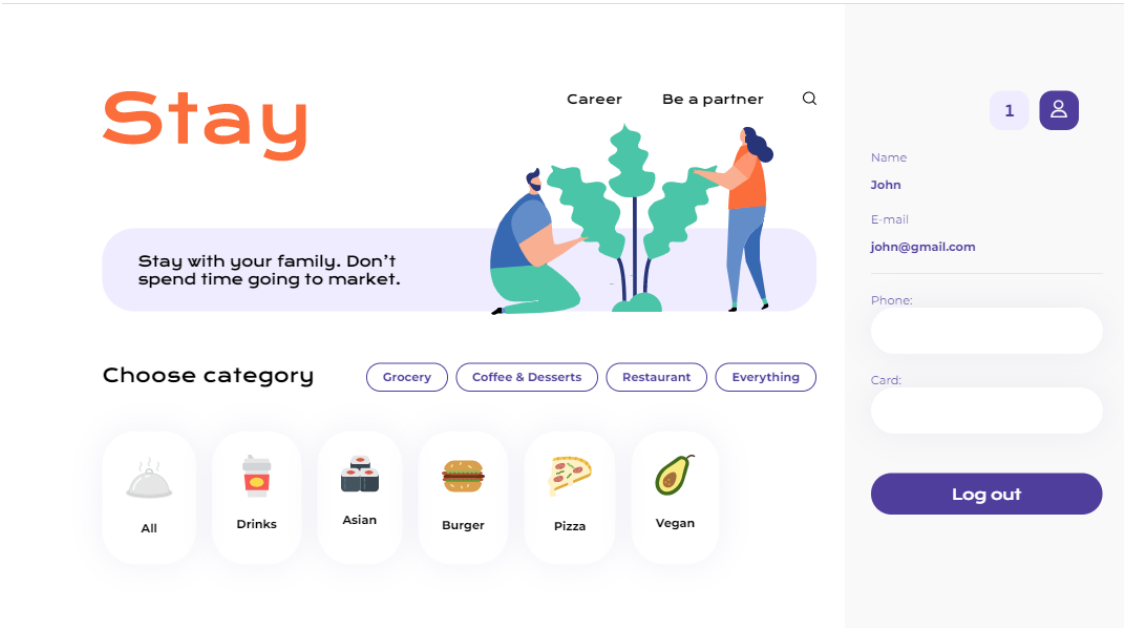


Рис.3.7.6. Перегляд акаунту користувача

Після натискання на кнопку Order:

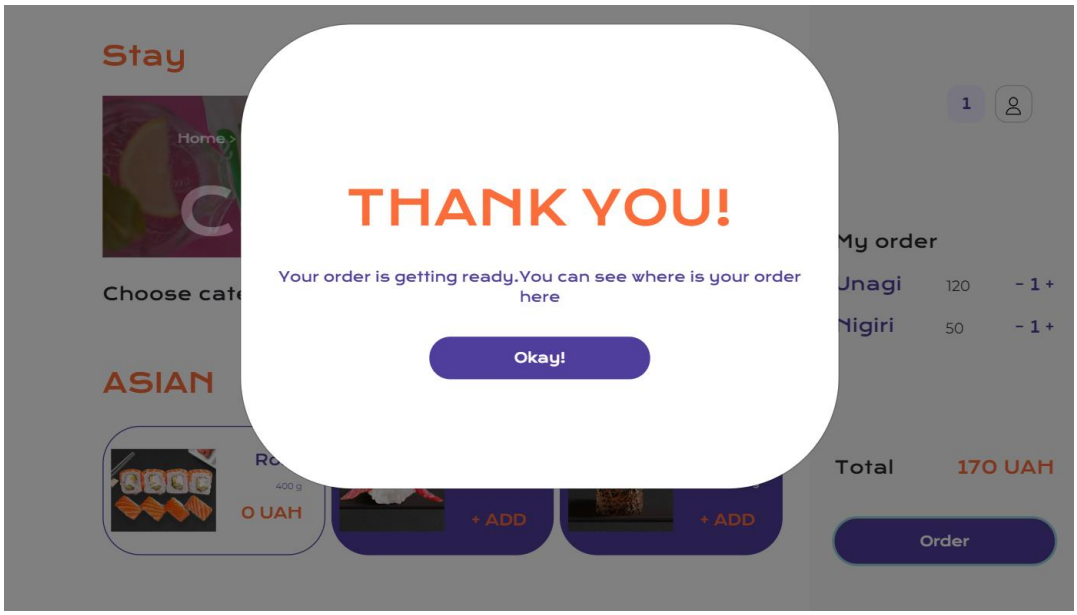


Рис.3.7.7. Завершення оформлення замовлення

Висновки

Отже, поставлена задача була виконана: розроблено веб-застосунок для організації доставки замовлень, який задовольняє потреби сучасного користувача та дозволяє швидко замовити доставку. Було проаналізовано конкурентів сайту, та визначено ключові функції, які потрібно мати сайту такого типу для повноцінної роботи. Розглянуті архітектурні рішення дозволили розбити програму на модулі, а аналіз типів сайтів – класифікувати застосунок.

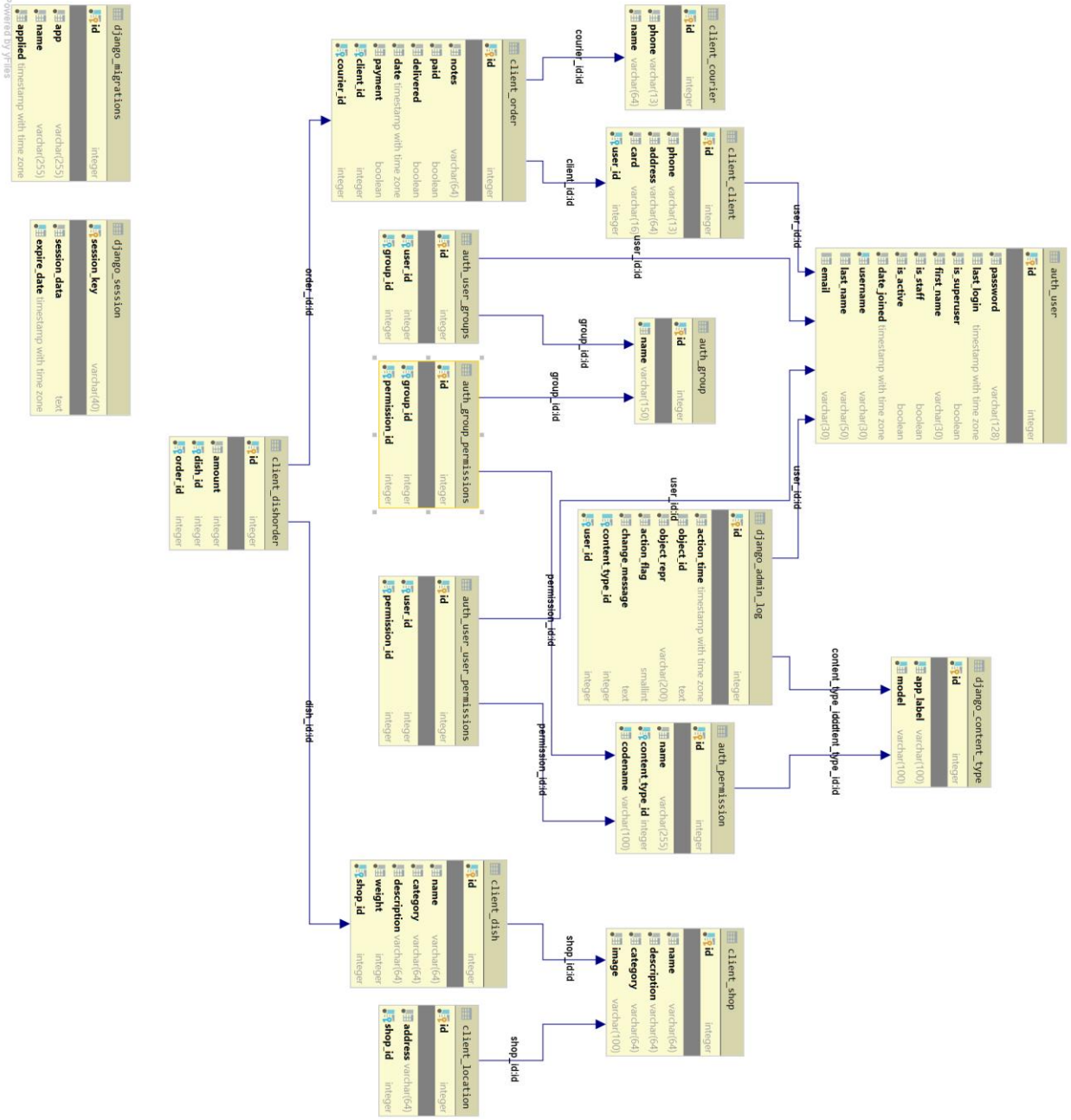
Обрані засоби розробки задовільнили технічні вимоги: середовище розробки надало можливість контролювати кожен процес, файл, та, звісно, базу даних, із одного інтерфейсу – вікна середовища розробки. А обрані мови та фреймворк дозволили комплексно підійти до вирішення поставлених задач розробки та реалізувати такий сервіс.

Розроблена система є веб-застосуванням для клієнтів сервісу доставки, та має напрямки для подальшого вдосконалення: створення інтерфейсів сайту для кур’єрів, щоб вони здійснювали доставку, та розширення функціоналу для клієнта – комунікація із кур’єром, наприклад, чат. Також важливим завданням є додавання пошуку за назвою, адже в системі присутній лише пошук за критерієм.

Список використаної літератури

1. Стаття з рейтингом сервісів доставки міста Києва [Електронний ресурс] https://blog.pokupon.ua/uk/prosto-privezi-mne-edy-top-10-servisov-dostavki-v-kieve-i-prijatnyj-bonus/
2. Стаття із відгуком журналіста на роботу сервісу Glovo [Електронний ресурс] https://retailers.ua/news/menedjment/8221-tri-kurera-i-tolko-odin-chas-na-dostavku-obzor-rabotyi-kurerskoy-slujbyi-glovo
3. Відео порівняння двох сервісів доставки [Електронний ресурс] https://www.youtube.com/watch?v=MFmWmtA0TxY
4. MVC-pattern [Електронний ресурс] https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm
5. Операції CRUD [Електронний ресурс] https://uk.wikipedia.org/wiki/CRUD
6. Таблиця співвідношень операцій CRUD [Електронний ресурс] https://en.wikipedia.org/wiki/Create,_read,_update_and_delete
7. Світова інтернет-статистика за 2020 рік [Електронний ресурс] https://www.websitehostingrating.com/internet-statistics-facts/
8. Стаття, у якій розглянуто класифікацію сайтів [Електронний ресурс] https://99designs.com/blog/web-digital/types-of-websites/
9. Python language [Електронний ресурс] https://www.python.org/
10. Django framework [Електронний ресурс] https://www.djangoproject.com/
11. PostgreSQL [Електронний ресурс] https://www.postgresql.org/
12. PyCharm IDE [Електронний ресурс] https://www.jetbrains.com/ru-ru/pycharm/

Додаток А. Схема бази даних



Додаток Б. Серверний код

Файл-контролер, який містить всі функції, які виконує система:

```
from django.http import HttpResponse, Http404, HttpResponseRedirect
from django.shortcuts import render, redirect
from django.contrib.auth.models import User, auth
from django.contrib import messages
from django.views.decorators.csrf import csrf_protect

from .models import Shop, Client, Dish, Order, DishOrder
from .forms import PhoneForm, CardForm

def index(request):
    context = {
        "shops": Shop.objects.all()
    }
    return render(request, "client/index.html", context)


def shop(request, shop_id):
    try:
        shop = Shop.objects.get(pk=shop_id)
    except Shop.DoesNotExist:
        raise Http404("Shop does no exist")
    context = {
        "shop": shop,
        "dishes": shop.dishes.all(),
        "shop_dish_cat": shop.dishes.all().values_list('category',
flat=True).distinct(),
    }
    if not request.user.is_anonymous:
        account_context = {
            "email": request.user.username,
            "name": request.user.first_name,
        }
        context.update(account_context)
    return render(request, "client/shop.html", context)


@csrf_protect
def first(request):

    context = {
        "shop_categories": Shop.objects.values_list('category',
flat=True).distinct(),
        "dish_categories": Dish.objects.values_list('category',
flat=True).distinct(),
        "shops": Shop.objects.all(),
        "asian_shops": Shop.objects.filter(dishes__category="Asian"),
        "burger_shops": Shop.objects.filter(dishes__category="Burger"),
        "pizza_shops": Shop.objects.filter(dishes__category="Pizza"),
        "vegan_shops": Shop.objects.filter(dishes__category="Vegan"),
        "drinks_shops": Shop.objects.filter(dishes__category="Drinks"),

    }

    if not request.user.is_anonymous:
        if request.method == 'POST':
```

```
            phone_form = PhoneForm(request.POST)
            card_form = CardForm(request.POST)

            if phone_form.is_valid():
                phone = phone_form.cleaned_data['phone']
                Client.objects.filter(user_id=request.user.id).update(phone=phone)
            if card_form.is_valid():
                card = card_form.cleaned_data['card']
                Client.objects.filter(user_id=request.user.id).update(card=card)

        else:
            phone_form = PhoneForm()
            card_form = CardForm()
        account_context = {
            "email": request.user.username,
            "name": request.user.first_name,
            "client": Client.objects.all().filter(user_id=request.user.id).first(),
            "phone_form": phone_form,
            "card_form": card_form,
        }
        context.update(account_context)

    return render(request, "client/first.html", context)


def register(request):
    if request.method == 'POST':
        first_name = request.POST['first_name']
        username = request.POST['username']
        password1 = request.POST['password1']

        if User.objects.filter(username=username).exists():
            messages.info(request, "Username taken")
            return redirect('/register')
        else:
            user = User.objects.create_user(username=username, password=password1,
                                            first_name=first_name)
            client = Client.objects.create(user_id=user.id)
            user.save()
            client.save()
            print("User created")
            return redirect('/login')
    else:
        return render(request, "client/register.html")


def login(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']

        user = auth.authenticate(username=username, password=password)

        if user is not None:
            auth.login(request, user)
            return redirect("/")
        else:
            messages.info(request, "Invalid credentials")
            return redirect('/login')
    else:
        return render(request, 'client/login.html')
```

```
def logout(request):
    auth.logout(request)
    return redirect('/')


def add_user_info(request):
    return render(request, "client/add_user_info.html")


def add_user_info_submit(request):
    if request.method == 'POST':
        phone = request.POST['phone']
        address = request.POST['address']
        card = request.POST['card']
        user = auth.get_user(request)
        client = Client.objects.create(phone=phone, address=address, card=card,
user=user)

        client.save()
        print("Client created")
        return redirect('/')
    else:
        return render(request, 'client/add_user_info.html')


def add_phone(request):
    if request.method == 'POST':
        user = auth.get_user(request)
        phone = request.POST['phone']
        client = Client.objects.create(phone=phone, user=user)
        client.save()
        print("Client created")
        return redirect('/')
    else:
        return redirect('/')
```

Додаток В. Клієнтський код

JavaScript-файл, який надає динамічності сторінці:

```
$(document).ready(function () {
    const cats = document.getElementsByClassName("dish-category");
    const shops0 = document.getElementsByClassName("shops")[0];
    const shops = document.getElementsByClassName("shops");
    const dishCard = document.getElementsByClassName("dish-card");

    const accountIconNotClicked = document.getElementById("accountNotClicked");
    const accountIconClicked = document.getElementById("accountClicked");
    const orderBlock = document.getElementById("orderBlock");
    const accountBlock = document.getElementById("accountBlock");
    const mainPage = document.getElementById("main-page");
    const logoSmall = document.getElementById("logoSmall");
    const logoBig = document.getElementById("logoBig");
    const btnAdd = document.getElementById("btnAdd");
    const btnAddToCart = document.getElementById("btnAddToCart");
    const myOrderContent = document.getElementById("my-order-content");
    const emptyOrderText = document.getElementById("empty-order-text");
    const submitOrder = document.getElementById("submitOrder");
    const orderPriceValue = document.getElementById("order-price-value");
    const addToCart = document.getElementsByClassName("add-to-cart");
    let totalSum = 0;

    for (const a of addToCart) {

        a.addEventListener('click', function (event) {
            emptyOrderText.style.display = "none";
            const dishIdAndPrice = $(this).attr('id');
            const words = dishIdAndPrice.split('+');
            const dishId = words[0];
            const dishPrice = words[1];
            let dishPriceInt = parseInt(dishPrice);

            totalSum += dishPriceInt;
            orderPriceValue.innerHTML = "" + totalSum;

            var div = document.createElement("div");
            div.innerHTML = "" + dishId;
            div.className = "dishInOrder";

            var divPr = document.createElement("div");
            divPr.className = "dishInOrderPr";
            var spanPr = document.createElement("span");
            spanPr.innerHTML = dishPrice;
            spanPr.className = "dishInOrderPrSpan";
            divPr.appendChild(spanPr);

            var divParent = document.createElement("div");
            divParent.className = "dishInOrderFlexbox";

            var count = document.createElement("div");
            count.className = "dishInOrderCount";
            var span = document.createElement("span");
            span.innerHTML = "- 1 +";
            span.className = "dishInOrderSpan";
```

```
            count.appendChild(span);

            divParent.appendChild(div);
            divParent.appendChild(divPr);
            divParent.appendChild(count);

            myOrderContent.appendChild(divParent);
            if (submitOrder.style.backgroundColor !== "#503E9D") {
                submitOrder.style.backgroundColor = "#503E9D";
                submitOrder.style.color = "#FFFFFF";
            }

        })
    }


    if (accountBlock !== null) {
        accountBlock.style.display = "none";
        $('#btnAdd').hide();
        $('input').keyup(function () {
            if ($.trim(this.value).length > 0)
                $('#btnAdd').show()
            else
                $('#btnAdd').hide()
        });
        $('#btnAdd2').hide();
        $('input').keyup(function () {
            if ($.trim(this.value).length > 0)
                $('#btnAdd2').show()
            else
                $('#btnAdd2').hide()
        });
    }

    for (let i = 0; i < shops.length; i++) {
        shops[i].style.display = "none";
    }

    for (const dc of dishCard) {

        dc.addEventListener('click', function (event) {
            if (dc.style.backgroundColor === "#503E9D") {
                dc.style.backgroundColor = "#FFFFFF";
            } else {
                const shopId = $(this).attr('id');
                var shop = $('#' + shopId);
                shop.addClass("active-card");
            }
        })
    }

    for (const button of cats) {
        button.addEventListener('click', function (event) {
            // TODO
            shops0.style.display = "none";
            console.log(document.getElementById($(this).attr('id') + "Shops"));
            document.getElementById($(this).attr('id') + "Shops").style.display =
"flex";
        })
    }
    accountIconNotClicked.onclick = function () {
```

```
        accountIconNotClicked.style.display = "none";
        accountIconClicked.style.display = "flex";
        orderBlock.style.display = "none";
        accountBlock.style.display = "block";
    }
    accountIconClicked.onclick = function () {
        accountIconClicked.style.display = "none";
        accountIconNotClicked.style.display = "flex";
        accountBlock.style.display = "none";
        orderBlock.style.display = "flex";
    }


});
```

Головна сторінка – first.html

```
<!DOCTYPE html>
{% load static %}
<html lang="en">
<head>
    <!-- Bootstrap CSS -->
    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
          integrity="sha384-
ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">

    <meta charset="UTF-8">
    <title>Stay</title>

    <link
href="https://fonts.googleapis.com/css2?family=Krona+One&family=Montserrat:wght@400;7
00&display=swap"
          rel="stylesheet">
    <link rel="stylesheet" href="{% static 'css/first.css' %}">

</head>
<body>

<div class="wrapper">
    {% block main %}
        <div class="section-main">
            <header>
                {% block logo %}
                    <div class="logo-big" id="logoBig">
                        <svg width="259" height="90" viewBox="0 0 259 90" fill="none"
                             xmlns="http://www.w3.org/2000/svg">
                        </svg>
                    </div>
                {% endblock %}
                <div class="header-nav" id="headerNav">
                    <a href="#">Career</a>
                    <a href="#">Be a partner</a>
                    <div class="search-button">
                        <svg width="17" height="17" viewBox="0 0 17 17" fill="none"
xmlns="http://www.w3.org/2000/svg">
                        </svg>
                    </div>
```

```
            </div>
        </header>
        <div class="main-page" id="main-page">
            {% block promo %}
                <div class="promo-block">
                    <div class="promo-text">
                        Stay with your family. Don’t spend time going to market.
                    </div>
                    <div class="promo-img">
                        <svg width="360" height="245" viewBox="0 0 360 245"
fill="none"
                             xmlns="http://www.w3.org/2000/svg">
                                                    </svg>
                    </div>
                </div>
            {% endblock %}
            {% block content %}
                <div class="shop-categories">
                    <div class="shop-categories-text">Choose category</div>
                    <div class="shop-categories-nav">
                        {% for category in shop_categories %}
                            <button class="shop-category-button"> {{ category
}}</button>
                        {% endfor %}
                    </div>
                </div>
                <div class="dish-categories">
                    <div class="dish-category">
                        <div class="dish-category-img">
                            <img src="{% static 'img/all.svg' %}" alt="">
                        </div>
                        <div class="dish-category-text">All</div>
                    </div>
                    {% for category in dish_categories %}
                        <div class="dish-category" id="{{ category }}">
                            <div class="dish-category-img">
                                {% with "img/"|add:category|add:".svg" as
template %}
                                    <img src="{% static template %}" alt="">
                                {% endwith %}
                            </div>
                            <div class="dish-category-text">{{ category }}</div>
                        </div>
                    {% endfor %}
                </div>

                <div class="shops" id="AsianShops">
                    {% for shop in asian_shops %}
                        <div class="shop" onclick="openShopPage1()">

                            <div class="shop-img">
                                <a href="shops/{{ shop.id }}">
                                    <img src="{{ shop.image.url }}" alt="image-{{
shop.name }}">
                                </a>
                            </div>
                            <div class="shop-content">
                                <div class="shop-name">
                                    {{ shop.name }}
                                </div>
                                <div class="delivery-price">
```

```
                                        <span>Delivery</span>
                                        <div class="delivery-price-number">40
UAH</div>
                                    </div>
                                </div>

                            </div>
                        {% endfor %}
                    </div>

                    <div class="shops" id="BurgerShops">
                            {% for shop in burger_shops %}
                            <div class="shop" onclick="openShopPage1()">

                                <div class="shop-img">
                                    <a href="shops/{{ shop.id }}">
                                        <img src="{{ shop.image.url }}" alt="image-{{
shop.name }}">
                                    </a>
                                </div>
                                <div class="shop-content">
                                    <div class="shop-name">
                                        {{ shop.name }}
                                    </div>
                                    <div class="delivery-price">
                                        <span>Delivery</span>
                                        <div class="delivery-price-number">40
UAH</div>
                                    </div>
                                </div>

                            </div>
                        {% endfor %}
                    </div>
                {% endblock %}
            </div>
            {# -- Shop page -- #}
            <div class="shop-page">
                <div class="shop-page-title">
                    {#        <img src="{%  %}" alt="">#}
                </div>
            </div>
        </div>
    {% endblock %}
    {% include 'client/sidebar.html' %}
</div>

<!-- Modal -->
<div id="myModal" class="modal fade" role="dialog">
    <div class="modal-dialog modal-lg">

        <!-- Modal content-->
        <div class="modal-content">
            <div class="modal-body">
                <div class="title">THANK YOU!</div>
                <div class="text">
                    Your order is getting ready. You can see where is your order here
                </div>
                <button class="okay">Okay!</button>
            </div>
        </div>
```

```
    </div>
</div>

<script src="https://code.jquery.com/jquery-3.4.1.min.js"
        integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo="
        crossorigin="anonymous"></script>
<script
src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"
        integrity="sha384-
UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
        crossorigin="anonymous"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"
        integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
        crossorigin="anonymous"></script>

<script src="{% static 'js/main.js' %}"></script>
</body>
</html>
```

Сторінка замовлення – sidebar.html

```
<div class="section-sidebar">
    {% if user.is_authenticated %}
                <div class="sidebar-header">
            <div class="count-orders">
                <span>1</span>
            </div>
            <div class="account accountNotClicked" id="accountNotClicked"
onclick="openAccount()">
                    <span>
                         <svg width="23" height="23" viewBox="0 0 23 23" fill="none"
xmlns="http://www.w3.org/2000/svg">
                    </svg>
                    </span>
            </div>
            <div class="account accountClicked" id="accountClicked"
onclick="closeAccount()">
                    <span>
                         <svg width="23" height="23" viewBox="0 0 23 23" fill="none"
xmlns="http://www.w3.org/2000/svg">
</svg>
                    </span>
            </div>
        </div>
    {% else %}
        <div class="sidebar-header">
            <div class="sign-up sidebar-header-button">
                <a href="register">Sign up</a>
            </div>
            <div class="login sidebar-header-button">
                <a href="login">Log in</a>
            </div>
        </div>
    {% endif %}
    <div class="order-block" id="orderBlock">
        <div class="my-order">
            My order
        </div>
```

```
        <div class="my-order-content" id="my-order-content">
            <div class="empty-order-text" id="empty-order-text">
                You haven’t picked up anything yet.
            </div>
        </div>
        <div>
            <ul>
                {% for foo in orderDishes %}
                 <li> {{ foo }}</li>
                {% endfor %}

            </ul>
        </div>
        <div class="order-footer">
            <div class="order-price">
                <div class="order-price-text">Total</div>
                <div class="order-price-number">
                    <span id="order-price-value">0</span>
                    UAH
                </div>
            </div>
            <div class="order-button">
                <button id="submitOrder" type="button" class="btn btn-info btn-lg"
data-toggle="modal"
                        data-target="#myModal">
                    Order
                </button>
            </div>
        </div>
    </div>


    {#    -- Account -- #}
    {% if user.is_authenticated %}
        <div class="account-block" id="accountBlock">
            <div class="account-component account-name">
                <label for="name">Name</label>
                <div id="name" class="account-value">
                    {{ name }}
                </div>
            </div>
            <div class="account-component account-email">
                <label for="email">E-mail</label>
                <div id="email" class="account-value">
                    {{ email }}
                </div>
            </div>

            {% if client.phone == "" %}
                <div class="account-input account-phone">
                    <form action="/" method="post">
                        {% csrf_token %}
                        {{ phone_form }}
                        <button id="btnAdd" class="account-add"
type="submit">Add</button>
                    </form>
                </div>
            {% else %}
                <div class="account-component">
                    <label for="phone">Phone</label>
                    <div id="phone" class="account-value">
```

```
                        {{ client.phone }}
                    </div>
                </div>
            {% endif %}

            {% if client.card == "" %}
                <div class="account-input account-card">
                    <form action="/" method="post">
                        {% csrf_token %}
                        {{ card_form }}
                        <button id="btnAdd2" class="account-add"
type="submit">Add</button>
                    </form>
                </div>
            {% else %}
                <div class="account-component">
                    <label for="card">Card</label>
                    <div id="card" class="account-value">
                        {{ client.card }}
                    </div>
                </div>
            {% endif %}

            <button class="account-logout"><a href="logout">Log out</a></button>
        </div>
    {% endif %}
</div>
```

Сторінка реєстрації:

```
<!DOCTYPE html>
{% load static %}
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Stay</title>

    <link
href="https://fonts.googleapis.com/css2?family=Krona+One&family=Montserrat:wght@400;7
00&display=swap"
          rel="stylesheet">
    <link rel="stylesheet" href="{% static 'css/registerAndLogin.css' %}">
</head>
<body>
<div class="wrapper">
    <div class="logo">
        <svg width="109" height="38" viewBox="0 0 109 38" fill="none"
xmlns="http://www.w3.org/2000/svg">
        </svg>
    </div>
    <div class="title">
        <div class="left">
            Sign Up
        </div>
        <div class="right">
            <a href="login">Log In</a>
        </div>
    </div>
    <div class="form-wrapper">
```

```
        <form action="register" method="post" id="register-form">
            {% csrf_token %}
            <input type="text" name="first_name" placeholder="Name"><br>
            <input type="email" name="username" placeholder="E-mail"><br>
            <input type="password" name="password1" placeholder="Password"><br>
            <button type="submit" form="register-form" value="Submit">Sign
Up</button>
        </form>

    </div>
    <div class="register-img">
        <svg width="396" height="383" viewBox="0 0 396 383" fill="none"
xmlns="http://www.w3.org/2000/svg">
        </svg>
    </div>
</div>

</body>
</html>
```