Безручко Михайло Андрійович, КНД-42

Розробка програми для забезпечення сервісу доставки замовлень індивідуальних та корпоративних споживачів

Координатор: Гніденко М.П.

ВСТУП

Обрана тема: “Розробка програми для забезпечення сервісу доставки замовлень індивідуальних та корпоративних споживачів” є як ніколи актуальною в час, коли бушує епідемія коронавірусу. Адже якщо вдасться зменшити кількість контактів між людьми на деякий час, хоча б для таких повсякденних задач як харчування, ми швидше зможемо подолати цю важку епідеміологічну кризу по всьому світові та повернутися до нормального життя з вільним пересуванням по країні та за її межами. Також, дана тема є актуальною навіть по тій причині, що приготування їжі займає достатньо багато часу і якщо спеціалісти будуть мати можливість скоротити марнування часу на ті сфери які їх цікавлять в меншій мірі та сконцентруватися на своїх основних задачах, як результат вони отримують куди більший прогрес у першочерговій сфері розвитку.

На даний момент подібні сервіси вже існують деякий час та достатньо сильно увійшли в наше повсякденне життя. Але що стосується забезпечення максимальної ізольованості та безпечної передачі товару від постачальника клієнту - тут, ми маємо деякі проблеми.

Перші служби доставки їжі в Україні з'явилися в середині 90-х років, після розпаду Радянського Союзу. Але тоді інтернет ще не був в широкому доступі, тому їжу на будинок замовляли найчастіше по телефону або при особистому відвідуванні ресторану. У 2000-х роках почала зароджуватися комбінована кухня - доставка піци, суші, ролів, бургерів з одного ресторану.

Зазвичай замовлення їжі додому чи в офіс здійснюється за допомогою телефонного дзвінка або через Інтернет. Замовлення їжі онлайн стало популярним в країнах, де розвинене використання мережі Інтернет. В останні роки Україна не відстає в цьому сенсі від самих просунутих країн. Замовити доставку їжі можна на сайті компанії, в мобільному додатку, в месенджерах або на сайтах-агрегаторах.

Разом з сервісами доставки розвивалася і упаковка для страв. Люди часто скаржилися, що замовлена їжа прибувала до них холодною, а упаковка була незручною. Тому необхідно було придумати спосіб, щоб доставити їжу гарячою і зручною для вживання.

Об'єктом дослідження є розробка програми для забезпечення сервісу доставки замовлень індивідуальних та корпоративних споживачів. Розробка такої програми буде сприяти поліпшенню та спрощенню доставки продуктів харчування та різноманітних страв безпосередньо індивідуально до дому та в офіси з можливим розкладом постійних поставок. У системі буде можливість створити розклад поставки їжі на основі вподобань та з урахуванням можливих алергій на ті чи інші продукти харчування.

Отже предметом дослідження є сама система забезпечення сервісу доставки замовлень індивідуальних та корпоративних споживачів.

Метою розробки даної системи є зменшення проблем викликаних неможливістю витрачати певну кількість часу на приготування їжі, створення робочої системи з можливістю замовляти та відстежувати свої замовлення з боку клієнта, а також створювати та оновлювати меню свого закладу для клієнтів зі сторони продавця.

В процесі дослідження вирішувалися наступні завдання:

1. Вибір, проектування та розробка бази даних
2. Розробка серверної частини для обробки даних
3. Розробка веб-інтерфейсу для взаємодії клієнта з системою

1 АНАЛІЗ ТЕХНОЛОГІЧНОГО ФУНДАМЕНТУ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА ЗАГАЛЬНА ПОСТАНОВКА ПРОБЛЕМИ

Для розробки програми, яка реалізує систему сервісу замовлень для індивідуальних та корпоративних клієнтів необхідно визначити на якій платформі буде розміщений сервіс та описати технології за допомогою яких буде проводитися розробка програмного забезпечення.

Визначити поняття наступних технологій для клієнтської частини:

1. HTML
2. CSS + препроцессори (sass)
3. JavaScript
4. Svelte
5. ReactJS
6. Redux

А також визначити поняття наступних технологій для серверної частини:

1. Http/Https
2. Ruby
3. Ruby on Rails

1.1 Докладніше про веб-додатки. Чоме саме веб-платформа програми обрана для розробки програмного забезпечення.

При підготовці розробки сервісу доставки, насамперед, необхідно обрати платформу на якій буде реалізована система. Веб-платформа є найбільш актуальним вибором через декілька факторів: немає необхідності в розробці програмного забезпечення безпосередньо на кожну із існуючих платформ та операційних систем; немає необхідності підтримки кожної з платформ; легкодоступність веб-платформи додатку через запуск сервісу прямо з браузеру; можливість легкого оновлення

версії системи без додаткових маніпуляцій з боку користувачів; можливість надійного централізованого зберігання даних на сервері;

1.2 Технології розробки інтерфейсу користувача

При взаємодії користувача як клієнта з постачальником, сервіс виступає в ролі посередника та потребує деякий інтерфейс для їх комфортної взаємодії. Посередником такого роду інтерфейсу може виступати клієнтська частина програми. Такий інтерфейс не потрібно встановлювати на комп’ютер локально а лишь ввести конкретну адресу в браузерній строці та перейти по ній. В послідуючих розділах роботи будуть розглянуті технології на яких будується клієнтська частина програми.

1.2.1 HTML

Документація вказує, що HTML являє собою стандартизовану мову розмітки документів, призначених для відображення у веб-браузері. Ця технологія є будівельними блоками для розмітки, вона визначає значення і структуру веб-контенту.

Гіпертекстова розмітка HTML визначає синтаксис та правила вживання спеціальних вбудованих в мову інструкцій - тегів, що не відтворюються браузером, але вказують йому, як треба відобразити вміст документа: зображення, текст та інші допоміжні види інформації. Крім того, ця мова дозволяє зробити документ інтерактивним за допомогою гіпертекстових посилань, які пов'язують його з документами на будь-якому комп'ютері, а також з іншими ресурсами Інтернету.

Теги - елементи розмітки в HTML які здебільшого легко розуміти і вживати, так як вони складені зі звичайних англійських слів, скорочень і позначень. Наприклад, теги і пропонують браузеру почати і закінчити відображення тексту між ними з певними властивостями. В

нашому випадку текст, який міститься у даному тезі буде з курсивним накресленням (italic).

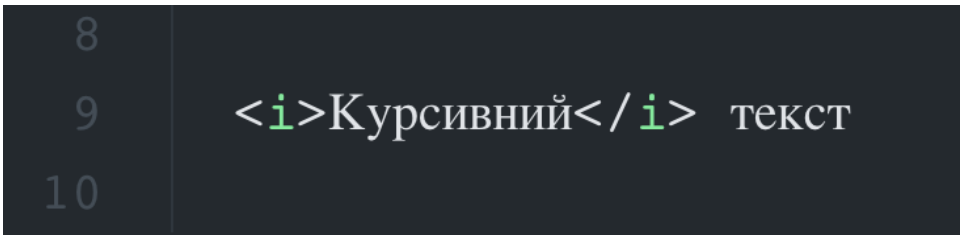

Рисунок 1.1 - Тег HTML

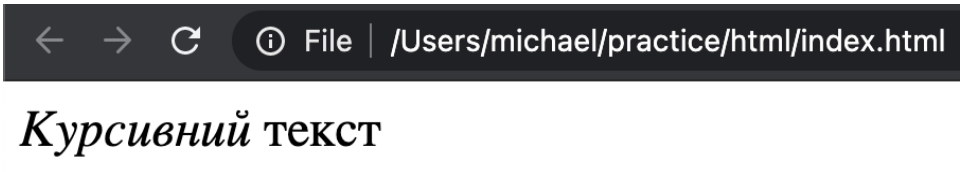

Рисунок 1.2 - Результат обернення тексту у тег

Кожен тег складається з імені тега, за яким може слідувати список атрибутів, при цьому все перераховане укладено між відкриваючим і закриваючим кутовими дужками (). Найпростіший тег це не що інше, як ім'я в дужках, наприклад <head> і <body>. Більш складні теги містять один або кілька атрибутів, що визначають або змінюють його дію. Відповідно до стандарту HTML ім'я і атрибути тега не чутливі до регістру. Немає ніякої різниці, між <main>, <Main>, <MAIN> або навіть <MaiN>, - всі вони еквівалентні.

Атрибути тега, якщо вони є, розташовуються слідом за ім'ям тега, розділені одним або декількома символами табуляції, пробілу або повернення каретки. Порядок, в якому записуються атрибути тега є байдужим.

Значення атрибуту поміщається за ім'ям атрибута через знак рівності (=). Можна вставляти пробіли навколо знака рівності, так що width = 6, width = 6 і width = 6 означають одне і те саме.

В HTML, якщо значенням атрибута є окреме слово або число (без пробілів), можна просто написати його після знака рівності. Всі інші значення повинні бути укладені в одинарні або подвійні лапки, особливо ті з них, що містять набір слів, розділених пробілами.

Окрім відкриваючих тегів які можуть нести деякі атрибути мається й закриваючі, які складаються тільки з імені тега, якому передує похила риска (/). Закриваючий тег не має атрибутів.

Тег може бути вкладений в область дії іншого тега, чим досягається складання їх дій на загальний сегмент документа.

У відповідності зі стандартами HTML вкладені теги слід завершувати, починаючи з самого внутрішнього і виконавши весь шлях у зворотному напрямку - за принципом «FILO».

FILO - абревіатура, що використовується в інформатиці для опису порядку, в якому здійснюється доступ до об’єктів. Також розшифровується як «першим увійшов, останнім вийшов».

Усі документи HTML повинні починатися з декларації типу документа <!DOCTYPE>. Декларація не є тегом HTML. Це інформація для браузера про те, який тип документа.

HTML документ має наступні обов'язкові теги:

<html></html> визначає початок і кінець HTML-документа.

<head></head> визначають початок і кінець заголовка документа. У заголовок документа зазвичай включається найменування документа і безліч додаткової службової інформації.

<title></title> теги для визначення найменування документа. Текст, поміщений між ними, сприймається браузером як назва документа і відображається ним у заголовку вікна.

<body></body> теги для визначення тіла HTML-документа. Тіло документа відповідає і за інформаційний зміст і за зовнішній вигляд інформації, представленої в вікні браузера.

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <title>Заголовок</title>
</head>
<body>

    <i>Курсивний</i> текст

</body>
</html>
```

Рисунок 1.3 - Стандартна розмітка HTML файлу

Саме ця стандартизована мова розмітки HTML буде використана для побудови макету клієнтської частини програми.

1.2.2 CSS

Каскадні таблиці стилів, які позначаються абревіатурою CSS, є простою мовою дизайну, призначеною для спрощення процесу розробки веб-сторінки з презентабельним зовнішнім виглядом.

CSS обробляє зовнішній вигляд частини веб-сторінки. За допомогою CSS можна керувати кольором тексту, стилем шрифтів, інтервалом між абзацами, розміром та розміщенням стовпців, фоновими зображеннями або кольорами, дизайном макета, варіаціями відображення для різних пристроїв та розмірами екрану (мається на увазі адаптивністю сторінок до різних пристроїв з яких можна користуватися клієнтською частиною), а також накладати безліч інших ефектів.

CSS забезпечує потужний контроль над поданням HTML-документа. Найчастіше CSS поєднується з мовами розмітки HTML.

Sass - препроцессор CSS, а також скриптова метамова яка підвищує рівень абстракції коду над стандартними каскадними стилями та спрощує файли CSS.

1.2.3 JavaScript

JavaScript (JS) - це легка, інтерпретуєма або оперативно компільована мова програмування з динамічною типізацією. Найбільш відома як мова сценаріїв для веб-сторінок. Вона використовується в багатьох середовищах, як на клієнтській стороні так і на не пов'язаних з браузером, наприклад, Node.js. Це заснована на прототипах, многопарадігмальна, однопотокова, динамічна мова, що підтримує об'єктно-орієнтований, імперативний і декларативний (функціональне програмування) стилі. Програмний код JavaScript може вбудовуватись в HTML-файли за допомогою тега, які було розглянуто нами у главі 1.2.2 <script>.

Динамічна типізація являє собою - мову програмування, яка основну частину перевірок типів виконує під час виконання програми, а не під час її компіляції. У динамічній типізації, значення мають типи, а змінні — ні, тому змінна може містити значення будь-якого типу.

За допомогою даної мови програмування реалізується логіка роботи веб-сторінки. Модальні вікна які будуть зроблені на сторінці, логіка роботи кнопок, інформативні вікна, а також обробка даних та перенаправлення на інші веб-сторінки буде написана саме на мові програмування JavaScript.

1.2.4 Svelte

Технологія з радикально новим підходом до створення користувацьких інтерфейсів. Відмінність даної технології від традиційних фреймворків, наприклад, таких як Vue, які виконують більшу частину своєї роботи в браузері є те, що Svelte переводить цю роботу на етап компіляції, який відбувається, коли тільки відбувається створення додатку. Замість використання таких методів, як віртуальне порівняння DOM, Svelte пише код, який тонким шляхом оновлює DOM при зміні стану інтерфейсу користувача.

DOM - розшифровується як “Об'єктна модель документа” являє собою міжплатформенний та незалежний від мови інтерфейс, оброблюючий HTML-документ як деревну структуру, де кожен вузол є об'єктом, що представляє частину документа. DOM представляє документ з логічним деревом. Кожна гілка дерева закінчується вузлом, а кожен вузол може містити об'єкти. Методи DOM дозволяють отримати програмний доступ до дерева. Також за допомогою них можна змінити структуру, стиль або зміст документа.

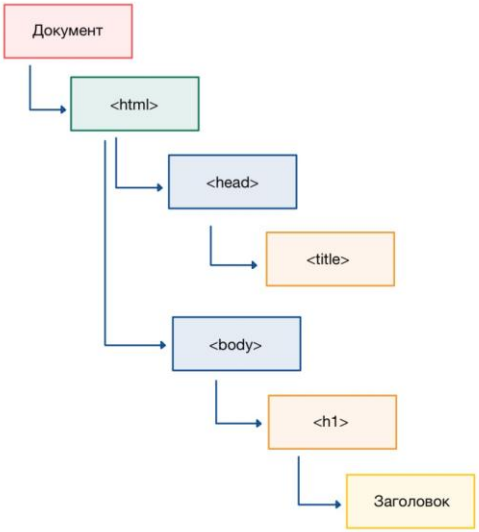

Рисунок 1.4 - Структура DOM дерева

За допомогою цієї технології буде реалізована реактивність клієнтської сторони програми.

1.2.5 ReactJS

ReactJS - бібліотека JavaScript з відкритим кодом, інтерфейс, для створення веб-інтерфейсів користувача або окремих його компонентів. Він розроблений та підтримується Facebook та спільнотою окремих розробників та компаній. React можна використовувати як основу при розробці односторінкових так званих SPA додатків.

SPA (Односторінковий додаток) - веб-додаток, який взаємодіє з користувачем шляхом динамічного перерендеру поточної веб-сторінки з новими даними з сервера, замість того, щоб веб-браузер за замовчуванням водночас завантажував цілі нові сторінки навіть з елементами які вже були завантажені на попередніх сторінках. Мета такого методу створення клієнтської частини веб-сайту є швидший перехід по сторінкам

прикладної програма, завдяки якому веб-сайт відчувається більше як рідна програма на комп'ютері.

У SPA весь необхідний код HTML, CSS та JavaScript отримуються браузером з першим завантаженням сторінки, або ж відповідні ресурси динамічно та поступово завантажуються та додаються на сторінку за необхідності, як правило, у відповідь на дії користувача. Після попереднього завантаження сторінки одного разу, сторінка більше ніколи не перезавантажується в будь-який момент процесу.

React займається лише управлінням станом і наданням цього стану в DOM, тому для створення програм React зазвичай використовуються додаткові бібліотеки для маршрутизації, а також певну функціональність на стороні клієнта.

1.2.6 Redux

Redux, як зазначено в документації, є передбачуваним контейнером стану для програм JavaScript. Це архітектура потоку даних програми, а не традиційна бібліотека або фреймворк. Завдяки даній технології при розробці програмного забезпечення системи замовлень індивідуальних та корпоративних клієнтів, буде змога керувати даними програми з більшою ефективністю у середовищі ReactJS.

1.3 Технології розробки серверної частини

1.3.1 Огляд серверної частини програми

Серверна частина веб-додатку описує способи взаємодії з собою, логіку обробки та передачі даних на клієнтську частину, взаємодію з базами даних, а також зі сторонніми програмами за допомогою програмного інтерфейсу(API).

API(Application Programming Interface) - абревіатура, яка являє собою програмний посередник, який дозволяє двом додаткам взаємодіяти один з одним.

За допомогою програмного інтерфейсу написаного на стороні серверної частини користувач сервісу, використовуючи інтерфейс на клієнтській частині, неявно для себе, буде взаємодіяти з API серверу. Сервер, в свою чергу, обробляти поступаючі запроси.

1.3.2 HTTP/HTTPS

HTTP - протокол передачі гіпертексту. HTTP протокол пропонує набір стандартів та правил, за якими визначається, що будь-яка інформація може бути передана у всесвітній павутині. HTTP надає стандартні правила для взаємодії веб-браузерів і серверів. Також HTTP є мережевим протоколом прикладного рівня, який побудований поверх TCP/IP моделі передачі даних.

TCP/IP це абстрактна мережева модель передачі даних, представлених в цифровому вигляді. Модель описує спосіб передачі даних від джерела інформації до одержувача та передбачає проходження інформації через чотири рівні, кожен з яких описується конкретним правилом яке зветься протоколом передачі даних. Набори правил, що вирішують завдання з передачі даних, складають стек протоколів передачі даних, на яких базується Інтернет.

Рисунок 1.5 - Рівні TCP/IP стеку

Протоколи прикладного рівня TCP/IP визначають організацію взаємодії різних мережевих комп'ютерів і форми подання інформації прикладних процесів за такої взаємодії. За ознаками взаємодії прикладних процесів виділяють два типи прикладного програмного забезпечення: програма-клієнт та програма-сервер.

HTTP при передачі даних використовує структурований текст гіпертексту, який встановлює логічний зв'язок між вузлами, що містять текст. Даний протокол не зберігає стан, оскільки кожна команда виконується окремо, без використання посилання на попередню команду запуску.

1.3.3 Ruby

Як і на стороні клієнтської частини, необхідно описувати логіку роботи інтерфейсу програми та логіку взаємодії його з сервером за

допомогою мови програмування JavaScript. Щодо сторони серверу, тут є куди більший вибір мов програмування. Для швидкої розробки була обрана мова програмування Ruby. Це динамічна мова програмування з відкритим кодом, орієнтована на простоту та продуктивність. Вона має елегантний синтаксис, який досить легко читати і писати. Це мова ретельного балансу. Вона вібрала в себе частини наступних мов програмування: Perl, Smalltalk, Eiffel, Ada та Lisp; та сформувала нову мову, яка збалансувала функціональне та імперативне програмування.

Рисунок 1.6 - Приклад програми яка виводить на екран заданий текст 5 разів

Можна бачити, що мова програмування Ruby створена досить інтуїтивно зрозумілою.

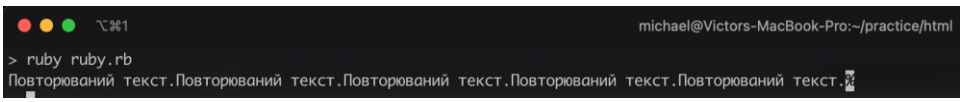

Рисунок 1.7 - Результат виводу програми

1.3.4 Model-View-Controller

Model-View-Controller(MVC) - у веб, використовується для розробки призначених для користувача інтерфейсів, характеризується тим, що поділяє логіку пов'язаної програми на три взаємопов'язаних елемента. Розроблявся для того, щоб відокремити внутрішнє представлення інформації від способів, якими інформація може надаватися користувачеві

і прийматися від нього. Шаблон дизайну MVC служить для відокремлення рівня презентації від бізнес-логіки.

MVC є популярним патерном у розробці додатків та веб-сайтів, і водночас один із найбільш широко використовуваних шаблонів дизайну програмного забезпечення для розробки програм та веб-програм. Шаблон дизайну контролера моделі перегляду має три наступних сегмента: модель, представлення, контроллер.

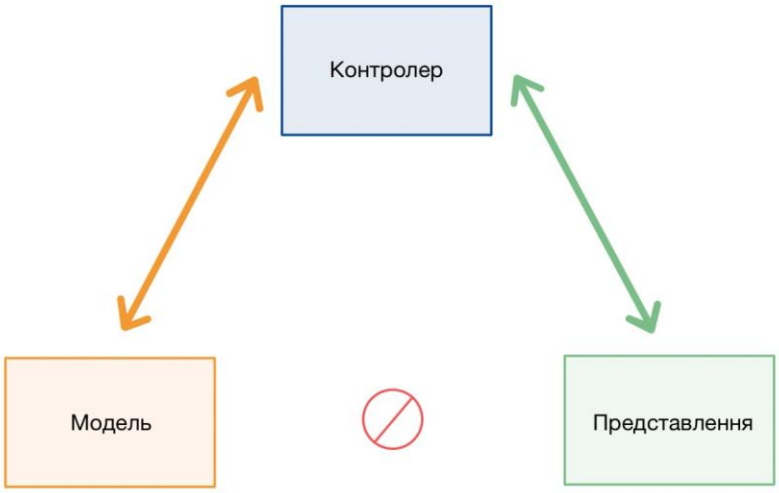

Рисунок 1.8 - Шаблон Модель-Представлення-Контролер

Представлення - частина програми, яка представляє подання даних. Створюються за даними, зібраними з даних моделі. Вимагає від моделі надання інформації, щоб повторно подавати користувачеві вихідну презентацію. Також, представлення представляє дані з чатів, діаграм та таблиць. Будь-яке представлення клієнта включатиме всі компоненти інтерфейсу користувача, такі як текстові поля, спадні меню тощо.

Контролер - частина програми, яка проводить обробку взаємодії користувача з сервером. Контролер інтерпретує всі вхідні дані від

користувача, інформуючи модель та вигляд, щоб змінити представлення за необхідністю. Команди контролера надсилають моделі інформацію для оновлення її стану. Також, контролер надсилає команди до пов'язаного з ним представлення для зміни презентації представлення (наприклад, прокручування певного документа або зміну тексту на сторінці).

Модель - компонент зберігаючий дані та пов'язану з ними логіку. Представляє дані, які передаються між компонентами контролера або будь-якою іншою пов'язаною бізнес-логікою. Наприклад, об’єкт контроллера отримає інформацію про клієнта з бази даних. Він обробляє дані та надсилає їх назад до бази даних або використовує їх для рендерінгу тих самих даних. Він відповідає на запит подань, а також відповідає інструкціям контролера щодо оновлення. Це є найнижчим рівнем шаблону, який відповідає за збереження даних.

1.3.5 Ruby on Rails

Ruby on Rails - веб-фреймворк, базується та написаний на мові програмування Ruby. Розроблений, щоб спростити програмування веб-додатків, роблячи припущення про те, що потрібно кожному розробнику для початку роботи. Завдяки вбудованим допоміжним засобам, дозволяє писати менше коду, виконуючи більше, ніж багато інших мов і фреймворків.

Реалізован за архітектурним шаблоном Model-View-Controller для веб-додатків, а також забезпечує їх інтеграцію з веб-сервером і сервером баз даних. Є відкритим програмним забезпеченням і розповсюджується під ліцензією MIT.

1.4 Доступ до надійних готових послуг на робочих місцях для підвищення робочої ефективності

1.4.1 Вступ

Останнім часом з'явилася досить стійка тенденція переміщення послуг з організації споживання продукції та обслуговування споживачів із залів ресторанного господарства до таких робочих місць як офіси та інших установ, місць відпочинку, а місць святкування ювілеїв та інших офіційних та неофіційних святкових подій - до домівок. В той же час, цю процедуру вдалося добре інтегрувати в цифровий вид. Така послуга має назву у міжнародній індустрії гостинності "catering".

Ця глава зосереджена на огляді різних літератур, що стосуються громадського харчування послуги та її роль у підвищенні ефективності роботи.

1.4.2 Сервісні служби громадського харчування

Кейтеринг - це діяльність з надання послуг харчування на віддаленій або такій ділянці, як готель, громадський будинок або інше місце. Організатор харчування забезпечує клієнтів гарячою або холодною їжею за вказаним часом та розташуванням. Їжа може складатися з гарячих страв для гурманів, приготованих на місці, або навіть у формі шведського столу. Служба харчування забезпечує їжу, напої та атмосферу для проведення заходів. Громадські організації забезпечують їжу вечірки, заходи та установи. Організатори харчування багато в чому пристосовуються до потреб своїх клієнтів. Вони можуть подати страви до 500 людей у банкетному залі або доставити ланчі в ресторані зустріч малого бізнесу. Вони можуть коригувати меню відповідно до дієтичних міркувань чи пропозицій обладнання (холодильні установки, виробники попкорну тощо) для прокату спеціальних заходів.

1.4.3 Працівники служби громадського харчування та порядок їх роботи

Ідеальна служба громадського харчування має своїх кухарів для приготування їжі, або вона може отримати їжу від підрядника або третьої сторони для доставки клієнту. У обідніх випадках, служба має офіціантів для приготування столів та подачі страв. Для фуршетів та на неформальних вечірках служба громадського харчування може направляти співробітників для приготування страв та подачі їх відвідувачам. До події, представник заходу допомагає клієнту вибрати їжу, місце проведення та прикраси в межах свого діапазону цін і встановлює спосіб виставлення рахунків. Клієнт переглядає пакет пропозицій із прогнозованими витратами. Для послуг громадського харчування, як правило, потрібно внести заставу до події.

1.4.4 Типи служб громадського харчування

Існує кілька видів послуг громадського харчування залежно від вимог клієнта щодо режиму роботи. Систематизація видів кейтерінгу за різними класифікаційними ознаками визначена у таблиці 1.4.1.

Таблиця 1.4.1 - Класифікаційні ознаки та види кейтерінгу

Класифікаційна ознака	Види кейтерінгу
За економічним змістом	соціальний
	комерційний
За формою перебування	виїзний
	стаціонарний
За типом надання послуг	тайм-кейтеринг

	плейс-кейтеринг
	мобільний кейтеринг
За статусом споживача послуги	VIP-кейтеринг
	мас-кейтеринг
За типом клієнтів	офісний кейтеринг
	готельний кейтеринг
	подійний-кейтеринг

Соціальний кейтеринг характеризується наявністю послуг з організації харчування для некомерційних організацій: притулків, лікарень, харчування військовослужбовців і співробітників, інших структур.

Комерційний кейтеринг означає надання послуг з організації харчування, що здійснюється з метою отримання прибутку.

Виїзний кейтерінг означає надання послуг з харчування на виїзд. Зокрема виїзний ресторан для будь якого свята може бути у вигляді офіційного прийому або фуршету для дачного ювілею. Організація фуршету на виїзді дозволить звільнитися від частини завдань, що супроводжують будь-який захід. Компанії, що організують виїзний кейтерінг, надають обладнання, кухню, надійний персонал тощо.

Стаціонарний передбачає, що підприємець володіє або орендує приміщення. Цей вид бізнесу застосовується при організації щоденного харчування підприємств, компаній, офісних центрів.

Тайм-кейтеринг – обслуговування заходу корпоративної вечірки, урочистого заходу, виставки, дилерської конференції. Даний вид кейтерингу передбачає разове надання послуг, орієнтоване на подію.

Плейс-кейтеринг – обслуговування віддалено розташованого підрозділу клієнта: доставка обідів в офіс корпоративного клієнта. Даний вид кейтерингу передбачає постійну чи контрактну роботу.

Мобільний кейтеринг – організації мобільного громадського харчування, широко відомі як вантажівки або візки з їжею, подорожують з місця в місто розміщують та подають бутерброди, напої, гамбургери та інші страви клієнтам у різних кварталах. Вони повинні мати ліцензію та перевіряти стан здоров'я, як і в ресторані. Тип пропонованої їжі та години роботи залежать від клієнтської бази громадського харчування. Мобільні вантажівки громадського харчування подають різноманітні обіди та закуски офісні та будівельні працівники у світлий час доби, тоді як візки з продуктами харчування обслуговують громадськість у цілому в районах з великим трафіком.

VIP-кейтеринг – надання послуг обслуговування високого рівня, що передбачає високо-класне обслуговування і якість харчування. Найчастіше це обслуговування невеликої групи або навіть одного клієнта, що передбачає виїзне ресторанне обслуговування з використанням найсучасніших кейтеринг технологій.

Офісний кейтеринг – організація харчування співробітників компаній.

Готельний кейтеринг – організація харчування постояльців готелів.

Подійний-кейтеринг – кейтеринг спеціальних подій.

1.4.5 Ідентифікація проблеми та передумови проблеми дослідження

Під час оцінки потреб було встановлено, що офісний персонал потребує обслуговування громадського харчування. Проблема відсутності послуг громадського харчування на місці змушує майже весь персонал виходити на обід і повертатися із запізненням, а також, можливо, мати

незадовільний досвід обідніх послуг. Співробітники вважають, що надання послуг громадського харчування на своєму робочому місці має великий вплив на їх ефективність з точки зору економії часу. Через те, що більша частина ресторанів у місті продає їжу за вищою ціною, постачання харчування через послуги доставки з ресторанів спричиняє економію грошей та отримання товару значно вищої якості. Зазвичай, робочий день персоналу починається з 8.00 ранку та мають обідню перерву на 1 годину з 13:00 до 14:00. Цей час є порівняно коротким, враховуючи відстань, яку вони повинні пройти в пошуках авторитетних ресторанів, а іноді й вистояти довгі черги, щоб отримати бажані страви. Іноді доводиться відкласти виконання якихось завдань, щоб вкластися у відведений на обід час, і це створює деякі робочі незручності.

1.4.6 Постановка проблеми

Правильне управління працівниками на робочому місці збільшує шанси бізнесу залишатися конкурентоспроможними на ринку. Одна з основних потреб працівників є харчування на робочому місці, яку вирішують послуги громадського харчування. Відбувається це тому, що організації мають свої офіси, які працюють з ранку до вечора, а доступні та якісні послуги громадського харчування розташовані далеко від робочого місця. Маючи зручну можливість замовлення послуг харчування в офісах, працівники мінімізують час, який буде витрачено на пошуки якісної та доступної послуги, і це насамперед позитивно впливає як на самопочуття самих працівників так і на виконання ними роботи. Частина робіт залишається зовсім незавершеною або виконаною неналежно, та іноді виникає психологічний стрес для персоналу, який отримує їжу в ненадійних місцях. В даний час якісна їжа та напої в офісах є дорогими, а іноді і дорогими і неякісними, а гігієнічні умови, де подають страви і зовсім можуть не співвідноситися з вимогами.

1.4.7 На кого впливає неможливість замовлення харчування

Вперш за все, є деякі співробітники, які не мають фінансових можливостей ходити в ресторани на обід оскільки у більшості ресторанів у центрі міста ціна їжі дуже висока. На організацію також впливає неефективність працівників, створена відсутністю такої послуги.

1.4.8 Вирішення проблеми

Організувати сервіс доставки харчів з будь якого типу ресторану різних цінових категорій з урахуванням уподобань та алергій користувача. Розробити платформу на якій користувачі можуть керувати своїми замовленнями з можливістю встановити розклад та адресу по якому буде надходити їжа. Бажано, щоб платформа була легкодоступна та без усіляких предустанов на персональний пристрій клієнта, тому чудовим рішенням буде розробити систему на основі веб-платформи. Таким чином, клієнт буде мати змогу досить швидко знаходити кухню яка буде йому до вподоби що входить до певної цінової категорії, а також, готуватися у спеціально обладнаних для цього місцях та відповідати гігієнічним нормам.

1.4.9 Висновок

Кейтеринг – це одна з гілок громадського харчування, принципом якої є віддалене надання послуг, тобто ресторанне обслуговування на виїзді. Сьогодні кейтеринг є одним із найбільш привабливих та затребуваних секторів ресторанного бізнесу в Україні, який потребує досконалого вивчення та популяризації.

2 РОЗРОБКА СЕРВЕРНОЇ ЧАСТИНИ ПРОЕКТУ

2.1 Створення проекту на Ruby on Rails

Для початку розробки серверного програмного забезпечення необхідно створити проект. Створити проект можна наступної командою: “*rails new*”. Обов’язковим параметром, після команди створення проекту йде ім’я проекту, дивитись на рисунок 2.1. Опціональним параметром є вибір системи управління базами даних. Для даного проекту вибір пав саме на PostgreSQL.

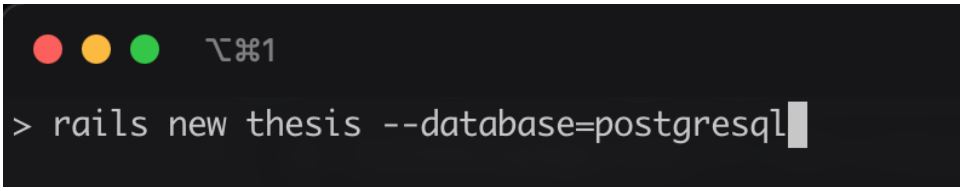

Рисунок 2.1 - Команда створення нового проекту з ім’ям “thesis” та системою управління базами даних PostgreSQL

Процес створення проекту та встановлення стандартних залежностей до нього можна спостерігати на рисунку 2.2.

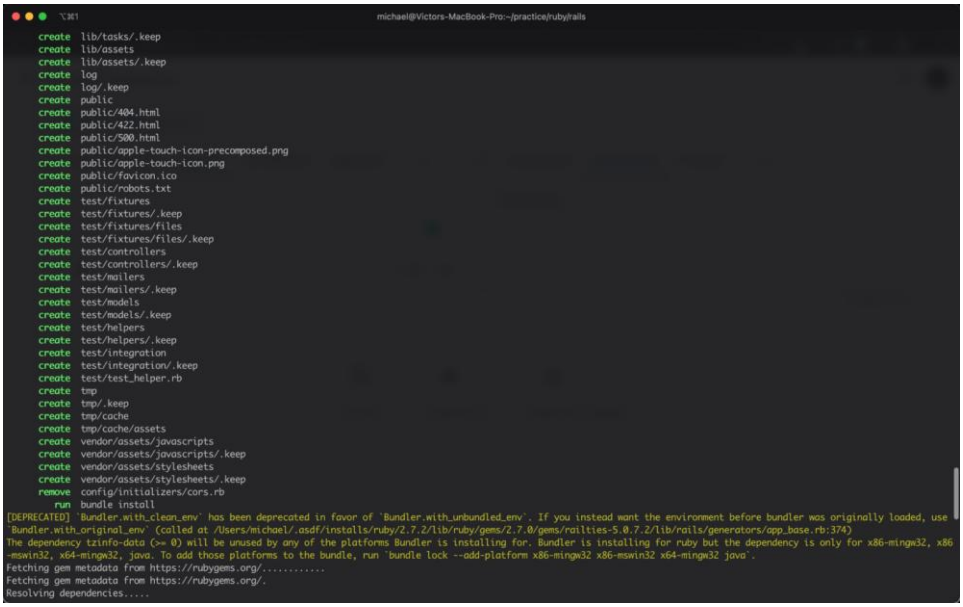

Рисунок 2.2 - Процес створення проекту та встановлення стандартних залежностей

Створення бази даних для проекту виконується наступною командою: “*rails db:create*”, що можна бачити на рисунку 2.3.

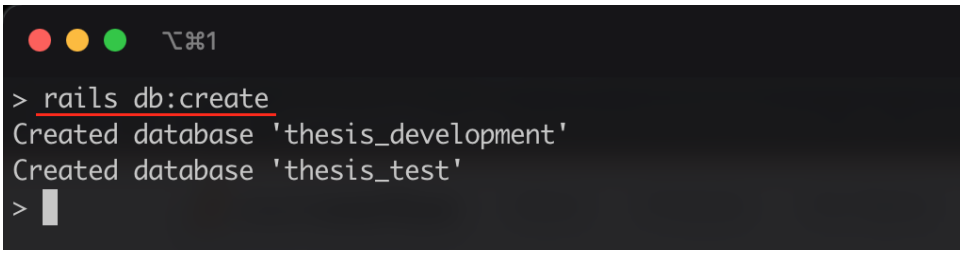

Рисунок 2.3 - Ініціалізація / створення бази даних

Задля перевірки створення бази даних за допомогою RoR команди, потрібно перейти через термінал під управління командного інтерфейсу PostgreSQL, як показано на рисунку 2.4.

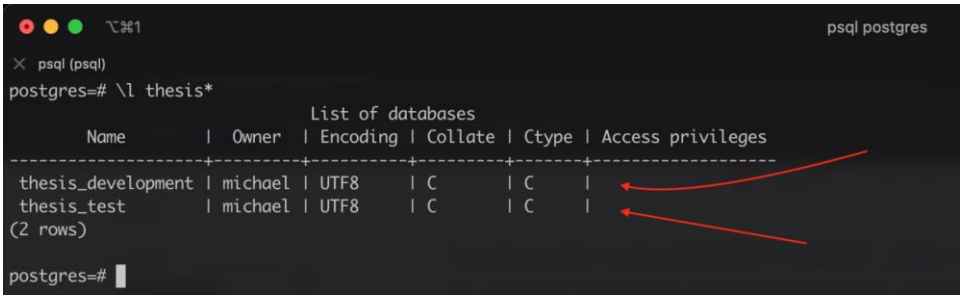

Рисунок 2.4 - Список створених таблиць для проекту “Thesis”

Можна бачити, що Ruby on Rails командою “*rails db:create”* автоматично створив декілька баз даних. Одну для тестування, а другу для розробки. Для перевірки працездатності створеного проекту потрібно його запустити. За стандартним запуском сервер буде піднятий на :3000 порту, що можна бачити на рисунку 2.5 та 2.6.

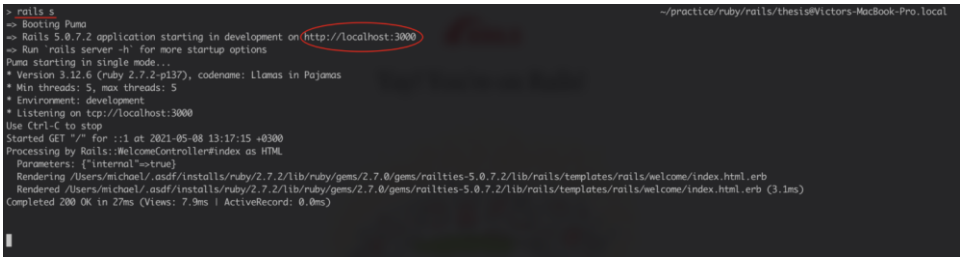

Рисунок 2.5 - Запуск локального сервера на порті :3000

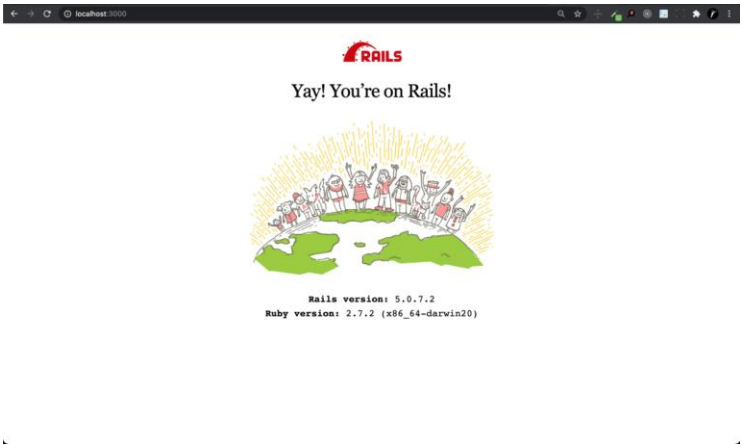

Рисунок 2.6 - Стартова сторінка перевірки працездатності серверу

2.2 Схематичний вигляд серверної частини системи

Серверна частина матиме наступну схематичну структуру зі відповідними зв'язками які можна бачити на рисунку 2.7.

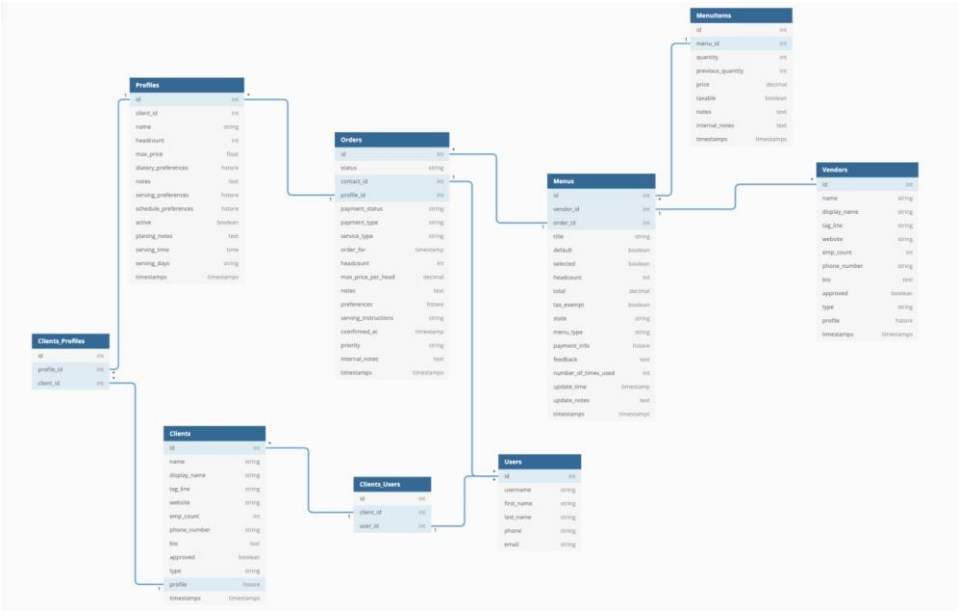

Рисунок 2.7 - Схема бази даних

2.3 Авторизація, ідентифікація, аутентифікація, куки, сесії та створення моделі User

Авторизація - процедура визначення прав доступу до ресурсів і управління цим доступом.

Ідентифікація - визначається як процедура, в результаті виконання якої для суб'єкта ідентифікації виявляється його ідентифікатор, який однозначно визначає цього суб'єкта в інформаційній системі.

Аутентифікація - є процедурою перевірки автентичності. Перевірка справжності користувача шляхом порівняння введеного ним пароля з паролем, збереженим в базі даних системи.

Кукі - це пари даних типу "ключ-значення", які зберігаються в браузері користувача до закінчення якогось певного терміну. Вони застосовні практично для будь-якого завдання, але найчастіше їх використовують, щоб зберегти користувача в тому ж місці веб-сторінки, якщо він втратить інтернет-з'єднання, або щоб зберігати прості настройки відображення сайту.

Сесії - за допомогою них браузери стежать, що користувач аутентифікований, навіть коли сторінка перезавантажується. HTTP запити не мають станів при переході між сторінками, тому завдяки сесіям є можливість відстежувати аутентифікований користувач чи ні. Ось чому важливі куки - вони дозволяють однозначно відстежувати користувача від запиту до запиту, поки не закінчиться їхній термін дії.

Завдяки можливостям фреймворку Ruby on Rails для написання серверної частини проекту, не потрібно буде хвилюватися про сесію користувача. Тому все що потрібно для створення таблиці “Користувачі” в базі даних, це створити модель з довільним ім’ям та полями для неї. Що стосується створення моделей, Ruby on Rails надає змогу створювати моделі прямо з термінального вікна (рисунок. 2.8).

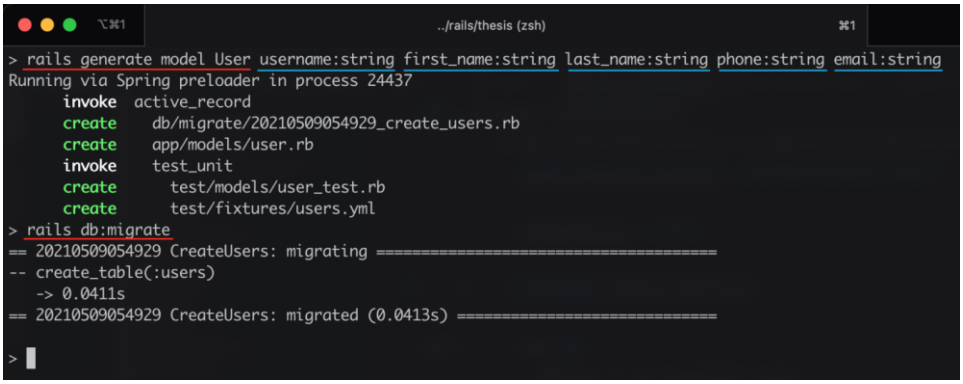

Рисунок 2.8 - Створення моделі User за допомогою Ruby on Rails та проведення міграції бази даних

Наступним шагом повинно бути створення контролерів для обробки логіки роботи даних та взаємодією з базою даних (рисунок. 2.9).

```ruby
def ajax_sign_up
  email = (params[:email] || (params[:user] && params[:user][:email])).downcase
  user = User.find_by_email(email)
  if user
    if user.deleted?
      SiteMailer.account_recovery(user).deliver_now
    else
      user.send_reset_password_instructions
    end
  else
    User.signup(email, office_id: get_default_office_id)
    if user
      user.confirmation_sent_at = nil
      user.send_confirmation_instructions
    end
  end

  render json: {}, status: 200
end
```

Блок коду виповниться, якщо користувач вже існує

В іншому разі спробувати зареєструвати нового користувача

Рисунок 2.9 - Контроллер для проведення реєстрації нових користувачів

На зображені контроллеру аутентифікації користувача можна бачити, що спочатку береться email та пароль користувача, який був введений в відведену форму на сайті. Наступним етапом є спроба ідентифікувати та авторизувати клієнта. Якщо email або пароль є неправильним, повертається помилка з повідомленням (рисунок. 2.10).

```ruby
def ajax_sign_in
  email = params[:email] || (params[:user]&& params[:user][:email])
  password = params[:password] || (params[:user]&& params[:user][:password])

  user =  User.find_first_for_ajax_signin(login: email)

  if user && user.valid_password?(password) && !user.deleted?
    sign_in(:user, user)
    @user = user
    render :me, status: 200
  else
    render json: {errors: ['The user / password you have entered is not recognized']},
           status: 422
  end
end
```

Рисунок 2.10 - Контроллер аутентифікації користувачів

2.4 Створення таблиці Clients в базі даних за допомогою RoR засобів

За допомогою засобу генерації моделі/таблиці в базі даних виконується створення нової моделі з назвою Client, яка абстрактно описує компанії, які потребують сервісу доставки їжі. Генерацію моделі Client можна бачити на рисунку 2.11.

Отформатировано: Украинский (Украина)

Отформатировано: Украинский (Украина)

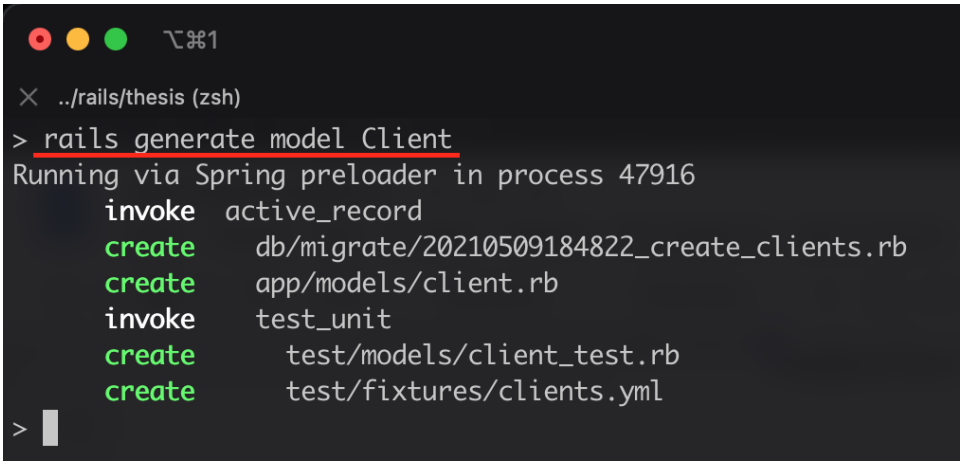

Рисунок 2.11 - Створення таблиці Clients

Для завершення етапу створення моделі Client, потрібно оприділити поля які буде містити в собі таблиця Clients в базі даних. Міграція буде виглядати наступним чином: оголошення класу “CreateClients” який успадковується від ActiveRecord модулю. ActiveRecord є стандартним модулем Ruby on Rails та працює за наступним принципом: “Об'єкт, який обертає рядок у таблиці або поданні бази даних, інкапсулює доступ до бази даних та додає логіку домену до цих даних”. Файл-міграцію можна спостерігати на рисунку 2.12.

```ruby
class CreateClients < ActiveRecord::Migration[4.2]
  def change
    create_table :companies do |t|
      t.string :name
      t.string :display_name
      t.string :tag_line
      t.string :website
      t.integer :emp_count
      t.string :phone_number
      t.text :bio
      t.boolean :approved
      t.string :type
      t.hstore :profile
      t.timestamps
    end
    add_index :companies, :name
  end
end
```

Рисунок 2.12 - Вигляд міграції при створенні таблиці Clients у базі даних

Після створення файлу-міграції, вона повинна бути затверджена наступною командою: “*rails db:migrate”*, як показано на рисунку 2.13.

Рисунок 2.13 - Процес міграції нової схеми бази даних

Можна бачити, що міграція виповнилась без помилок, та успішно інтегрувала зміни до старої схеми бази даних. Отже, на даному етапі, в системі існує вже дві таблиці, це Users і Clients (рисунок 2.14, рисунок 2.15).

2.5 Створення зв'язку між таблицею Users та Clients

```
class User < ApplicationRecord

  # associations
  has_many :clients
```

Рисунок 2.14 - Створення зв'язку User моделі з моделлю Client

```
class Client < Company

  # associations
  has_many :user_lists
```

Рисунок 2.15 - Зворотній зв'язок Client та User моделей

Зв’язок між таблицями Users та Clients, реалізований через посередницьку таблицю Clients_Users та схематично буде виглядати наступним чином (рисунок 2.16).

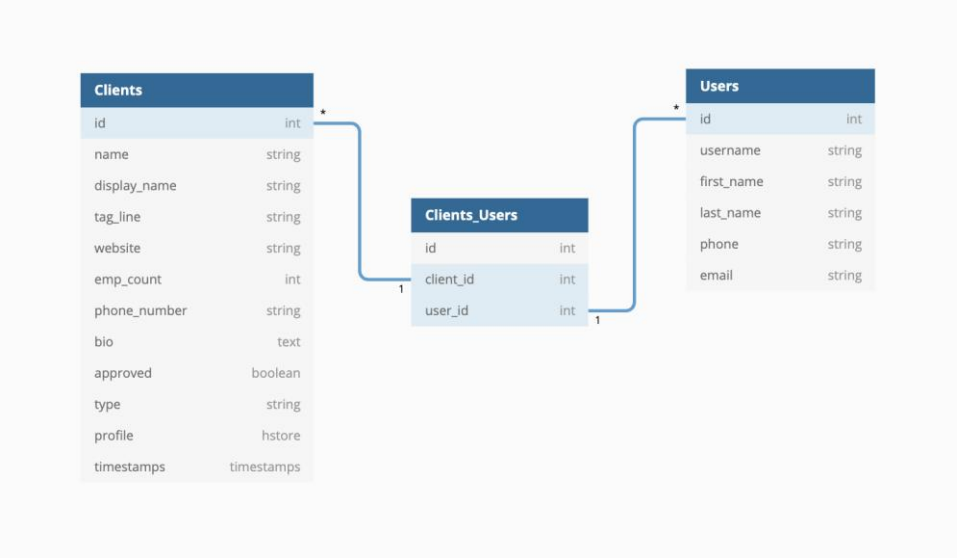

Рисунок 2.16 - Зворотній зв'язок Client та User моделей

В даному випадку, використання зв'язку “багато до багатьох” є найбільш оптимальним та логічним рішенням бо ж в компанії може працювати один і більше працівників. В той же час, один працівник може числитися у декількох компаніях одночасно. Реалізація даного зв'язку робиться завдяки вбудованому “магічному” методу *has_many*, який означає створення зв'язку “багато до багатьох” між моделями User та Client. Після визову метода “*has_many*” повинно йти ім'я таблиці через двокрапку.

Еквівалент на чистій мові SQL буде виглядати наступним чином:

```
CREATE TABLE client_user
(
   id integer NOT NULL,
   client_id integer NOT NULL,
   user_id integer  NOT NULL,
   PRIMARY KEY (client_id , user_id ),
   FOREIGN KEY (client_id) REFERENCES clients,
```

```
    FOREIGN KEY (user_id ) REFERENCES users,
);
```

Таким чином буде реалізован зв'язок моделей. Завдяки новому зв'язку є змога дістати дані по асоціаціям. Наприклад, команда “*Client.users*” дасть змогу вибрати всіх користувачів зв'язаних с конкретною компанією. Для взаємодії, а саме для створення, оновлення та видалення записів необхідні оброблюючі запроси до серверу контроллери. Зовнішній вигляд контролеру можна бачити на рисунку 2.17.

```ruby
def create_client_user
  Clients::Users::CreateUser
  .(input: params, who_done_it: current_user)
  .on_error   {|s| render json: {errors: s.errors.full_messages}, status: :unprocessable_entity }
  .on_success {|s| @client_user = s.user; @client = s.client; render :client_user }
end
```

Рисунок 2.17 - Контроллер створення нового клієнту

Створення нового клієнта та обробка інформації про нього досить велика, тому винесення основного коду до окремого модулю є очевидним рішенням розділення коду. Такі, окремо винесені модулі з кодом називають сервісами. Сервіси є модулями мови програмування Ruby та надають змогу робити будь-яку вкладеність а також імпортувати їх до файлів задля того щоб залишати основну гілку коду чистою. Готовий модуль зображений на рисунку 2.18.

```ruby
module Clients
  module Users
    class CreateUser

      attr_accessor :attrs,
                    :client,
                    :input,
                    :user,
                    :who_done_it

      def call(input:, who_done_it:)
        self.input          = input
        self.who_done_it    = who_done_it
        self.attrs          = input[:user]
        self.client         = Client.find_by_guid(attrs[:client_guid])
        self.user           = User.new

        _validate(attrs)
        _assign_attributes(user: user, attrs: attrs)
```

Рисунок 2.18 - Внутрішній вигляд сервісу Clients::Users::CreateUser

Логіка роботи сервісу наступна: прийняти на вхід параметри які прийшли до контроллеру з клієнтської частини програми, а потім були передані насамперед до сервісу; провести валідацію полей задля коректності даних та для перевірки їх існування в цілому; назначити атрибути відповідно полям та записати до бази даних. Функція call буде автоматично визвана як початкова точка роботи сервісу та приймає на вхід деякі параметри як і звичайна функція. Валідація проводиться також в окремій функції у сервісі Clients::Users::CreateUser. Валідацію відповідних полів можна бачити на рисунку 2.19.

```ruby
def _validate(attrs)
  _validate_email(attrs[:email])

  errors.add(:first_name, 'should not be blank') if attrs[:first_name].blank?
  errors.add(:last_name, 'should not be blank')  if attrs[:last_name].blank?
  error! if errors.any?
end

def _validate_email(email)
  return errors.add(:email, "can not be blank") if email.blank?
  return errors.add(:email, "has wrong format") if !(email =~ Devise.email_regexp)
  errors.add(:email, "already taken") if User.where(email: email.downcase).count > 0
end
```

Рисунок 2.19 - Валідація Clients::Users::CreateUser сервісу

Якщо одне з полів не пройде валідацію, буде повернена помилка та робота сервісу припиниться. Співвідношення даних до конкретних полів показано на рисунку 2.20.

```ruby
def _assign_attributes user:, attrs:
  user.email                    = attrs[:email].to_s.downcase
  user.first_name               = attrs[:first_name]
  user.last_name                = attrs[:last_name]
  user.user_list_ids            = attrs[:user_list_ids] || []
  user.neo_allergy_preferences  = attrs[:neo_allergy_preferences]
  user.neo_dietary_preferences  = attrs[:neo_dietary_preferences]
  user.notes                    = attrs[:notes]

  user.email_subscription         = attrs[:email_subscription] || false
  user.receives_employee_feedback = attrs[:receives_employee_feedback] || false

  user.save!(validate: false)
end
```

Рисунок 2.20 - Призначення атрибутів відповідним до них полям

По-аналогії робляться й контролери для оновлення та видалення записів. Також за допомогою сервісів визваних у контролері. Для того щоб

Отформатировано: Украинский (Украина)
Отформатировано: Украинский (Украина)
Отформатировано: Украинский (Украина)
Отформатировано: Украинский (Украина)
Отформатировано: Украинский (Украина)
Отформатировано: Украинский (Украина)
Отформатировано: Украинский (Украина)

оновити або видалити записи клієнтів зазвичай необхідно створити окремий контроллер з роутами за якими буде проходити звернення до серверної частини клієнту. Контролери з відповідною логікою роботи можна бачити на рисунках 2.21 та 2.22 відповідно.

```
def update
  Clients::UpdateUser
  .(who_done_it: current_user, user_id: params[:id], params: params)
  .on_success{ |s| @user = s.user; render :show }
  .on_error{ |s| render json: {errors: s.errors.full_messages}, status: :unprocessable_entity }
end
```

Рисунок 2.21 - Контроллер оновлення клієнту

```
def destroy
  Clients::Users::DeleteUser
  .(input: params, who_done_it: current_user)
  .on_error   {|s| render json: {errors: s.errors.full_messages}, status: :unprocessable_entity }
  .on_success {|r| render json: {}, status: :ok }
end
```

Рисунок 2.22 - Контроллер видалення клієнту

2.6 Створення моделей Vendor, Menu та MenuItem

Для розділення компаній-клієнтів та компаній-постачальників є необхідність в створенні додаткової таблиці Vendors з записами усіх компаній-постачальників. Саме таким чином не буде порушено нормалізацію бази даних. Модель Vendor має всі ті й самі поля як і модель Client, тому буде успадкувана саме від неї (рисунок 2.23).

```
class Vendor < Company

  has_many :menus
  has_many :set_menus
  has_many :orders, -> { where(menus: {selected: true}) } , through: :menus
```

Рисунок 2.23 - Модель Vendor

У кожного постачальника має бути змога додати своє меню з відповідними стравами, тож для реалізації цього необхідно створити ще дві таблиці Menu та MenuItems а також зв'язок між ними (рисунок 2.24).

```ruby
class CreateMenus < ActiveRecord::Migration[4.2]
  def change
    create_table :menus do |t|
      t.integer :vendor_id
      t.integer :order_id
      t.string :title
      t.boolean :default
      t.boolean :selected
      t.integer :headcount
      t.decimal :total, precision: 8, scale: 2
      t.boolean :tax_exempt
      t.string :state
      t.string :menu_type
      t.hstore :payment_info
      t.text :feedback
      t.integer :number_of_times_used
      t.timestamp :update_time
      t.text :update_notes
      t.timestamps
    end
  end
end
```

Рисунок 2.24 - Файл-міграції створення таблиці Menus

Як і завжди, у файлі-міграції вказуються назви полів та їх типи, які будуть належати до створюваної таблиці. За аналогією створюється й наступна таблиця MenuItems (рисунок 2.25).

```ruby
class CreateMenuItems < ActiveRecord::Migration[4.2]
  def change
    create_table :menu_items do |t|
      t.integer :menu_id
      t.integer :item_id
      t.integer :quantity
      t.integer :previous_quantity
      t.decimal :price, precision: 10, scale: 2
      t.boolean :taxable
      t.text :notes
      t.text :internal_notes
      t.timestamps
    end
  end
end
```

Рисунок 2.25 - Файл-міграції створення таблиці MenuItems

Завжди, після створення файлу міграції потрібно затвердити зміни схеми бази даних (рисунок 2.26).

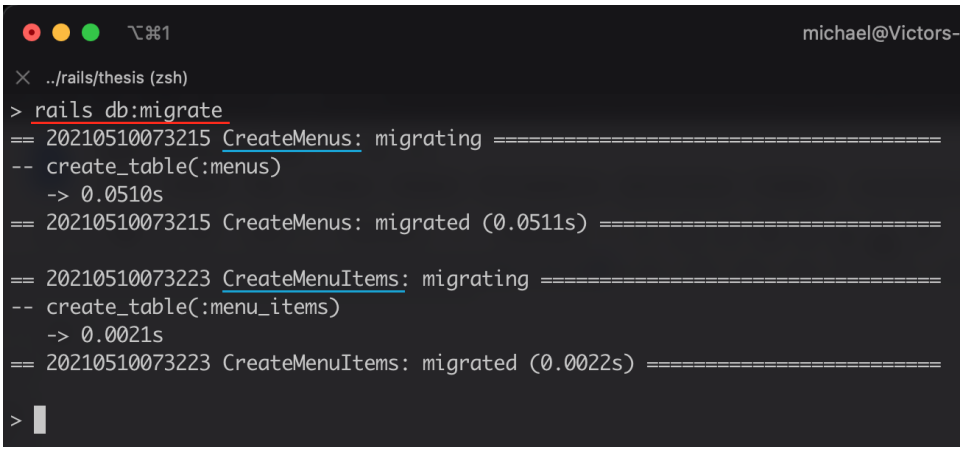

Рисунок 2.26 - Проведення міграції нової схеми бази даних

Схематичну складову між моделями Vendor, Menus та MenuItem можна бачити на рисунку 2.27.

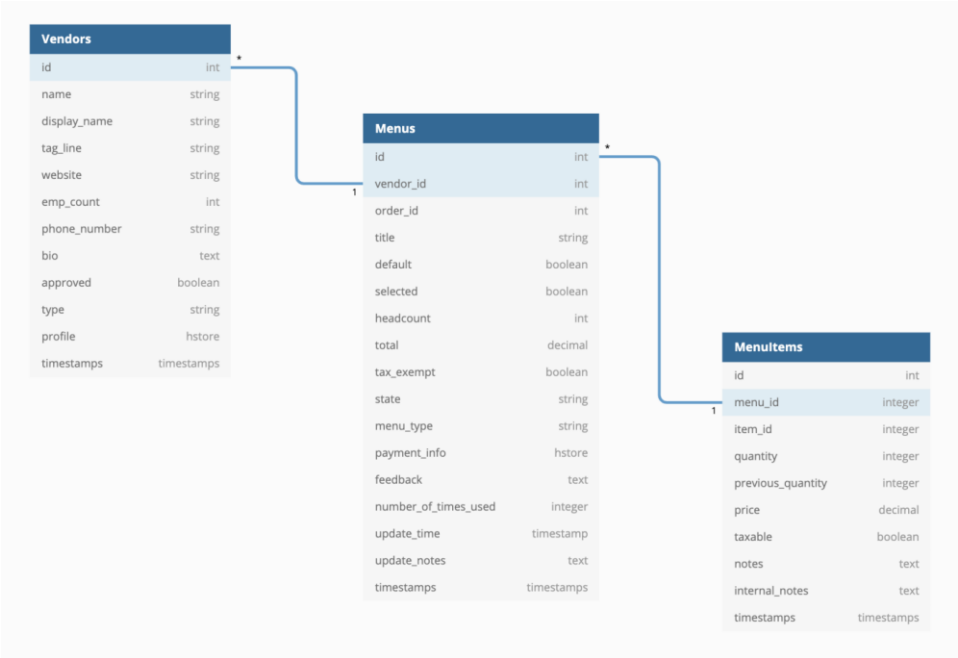

Рисунок 2.27 - Проведення міграції нової схеми бази даних

2.7 Створення моделі Profile для налаштувань заказів

Модель Profile відображає функцію налаштування заказів. В системі буде можливість налаштовувати окремі елементи замовлень. На основі вибраного типу профайлу, буде реалізована змога створювати замовлення конерктних типів, будь-то для індивідуальних чи корпоративних клієнтів. Це буде дещо змінювати логіку роботи замовлень. Файл-міграції налаштувань зображено на рисунку 2.28.

```ruby
class CreateProfiles < ActiveRecord::Migration[4.2]
  def change
    create_table :profiles do |t|
      t.integer :client_id
      t.string :name
      t.integer :headcount
      t.float :max_price
      t.hstore :diatery_preferences
      t.text :notes
      t.hstore :serving_preferences
      t.hstore :schedule_preferences
      t.boolean :active
      t.text :planing_notes
      t.time :serving_time
      t.string :serving_days
      t.boolean :is_backup_plan
      t.boolean :is_notification_sent
      t.datetime :backup_plan_enabled_at
      t.datetime :backup_plan_disabled_at
      t.timestamps
    end
  end
end
```

Рисунок 2.28 - Файл-міграції моделі Profile

Типи профайлів створюються за допомогою конструкції ENUM у MySQL. Тип даних ENUM у MySQL є рядковим об'єктом. Це дозволяє обмежити значення, вибране зі списку дозволених значень під час створення таблиці. Це означає, що кожен стовпець може мати одне із зазначених можливих значень. У середовищі Ruby on Rails, сам рядковий об’єкт ENUM буде створений за допомогою типу даних “Кортеж”. Це звичайний массив, до якого заборонено вносити будь-які зміни. Модуль, описуючий існуючі типи профайлу можна побачити на рисунку 2.29.

```ruby
class Profile
  module MealExperiences
    GROUP_ORDERING = 'Group Ordering'
    HOME_DELIVERY = 'Home Delivery'

    ALL = [
      GROUP_ORDERING,
      HOME_DELIVERY,
    ].freeze

    DICT = {
      group_ordering: GROUP_ORDERING,
      home_delivery: HOME_DELIVERY,
    }.freeze
  end
end
```

Рисунок 2.29 - Опис різних типів профайлу

2.8 Створення системи планування замовлень

Для зручного створення шаблонних замовлень та взагалі користування сервісом, є необхідним створення системи планування замовлень з урахуванням вподобань клієнта. Описану модель можна бачити на рисунку 2.30.

```ruby
class UserBackupPlanPreference < ApplicationRecord
  module DietaryTags
    OMNIVORE    = 'omnivore'
    VEGETARIAN  = 'vegetarian'
    VEGAN       = 'vegan'
    GLUTEN_FREE = 'gluten_free'
    DAIRY_FREE  = 'dairy_free'

    ALL = [OMNIVORE, VEGETARIAN, VEGAN, GLUTEN_FREE, DAIRY_FREE].freeze
    SET_MENU_TAGS = {
      "omnivore" => 'Omnivore',
      "vegetarian" => 'Vegetarian',
      "vegan" => 'Vegan',
      "gluten_free" => 'Gluten-Free',
      "dairy_free" =>  'Dairy-Free',
    }.freeze
  end

  belongs_to :user
  belongs_to :profile

  validates :user, presence: true
  validates :profile, presence: true
  validates :schedule_preference, presence: true
end
```

Рисунок 2.30 - Опис моделі BackupPlan

Можна бачити, що зазначено декілька типів алергенів, а саме: “omnivore - вживаючий все, vegetarian - вегетаріанець, vegan - веган, gluten free - не вживаючий їжу без глютену, diary free - не вживаючий молочні продукти”. Завдяки зазначеному списку, користувач має можливість зробити відмітку про алергію на деякі продукти, а також вказати свої уподобання до страв у себе в налаштуваннях в кабінеті користувача.

2.9 Створення моделі Order

Файл міграції Order описує із яких параметрів буде складатися замовлення. Екземпляри класу Order є насамперед об’єктами цього класу та вже готовими, створеними записами у базі даних з відповідною інформацію в них. Для замовлення важливо мати інформацію про статус для відстеження етапу його обробки. Окрім статусу, замовлення може створюватись із зазначеною кількістю страв. Саме по цій причині, в таблиці Orders буде створено поле з назвою “headcount” куди буде вписана кількість людей для яких створюється замовлення. Кожне замовлення потрібно проводити через оплату. Для цього створюється ще декілька полів payment_type та payment_status. Детальну інформацію про поля у таблиці, можна бачити на рисунку 2.31.

```
class CreateOrders < ActiveRecord::Migration[4.2]
  def change
    create_table :orders do |t|
      t.string :status
      t.string :payment_status
      t.string :payment_type
      t.integer :contact_id
      t.string :service_type
      t.timestamp :order_for
      t.integer :headcount
      t.decimal :max_price_per_head, precision: 5, scale: 2
      t.text :notes
      t.hstore :preferences
      t.string :serving_instructions, :array => true
      t.integer :profile_id
      t.timestamp :comfirmed_at
      t.string :priority
      t.text :internal_notes
      t.timestamps
    end
  end
end
```

Рисунок 2.31 - Файл-міграції моделі Order

Схематичний зв’язок обидвох моделей Profile та Order буде зазначено на рисунку 2.32 нижче. Зв’язок реалізован за допомогою звичайного зв’язку

типу “Один до багатьох”, тому кожен запис замовлення у базі даних буде мати зовнішній ключ налаштувань, (зазвичай це ID) та посилатися на запис з конкретними налаштуваннями до нього.

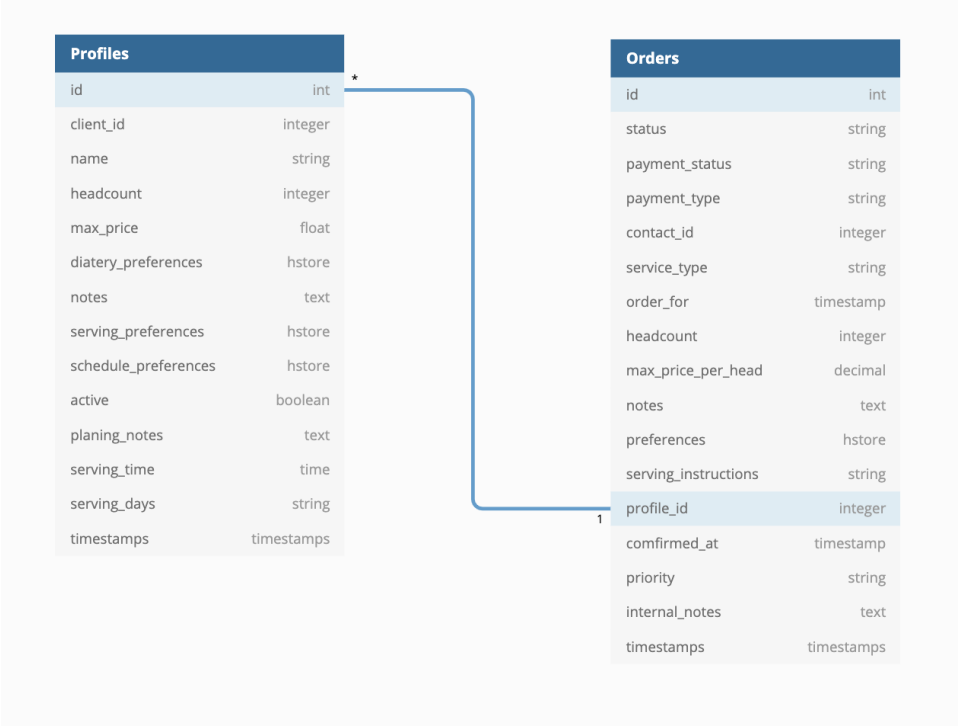

Рисунок 2.32 - Схематичний зв’язок моделей Profile та Order

3 РОЗРОБКА КЛІЄНТСЬКОЇ ЧАСТИНИ ПРОЕКТУ

3.1 Вигляд інтерфейсу списку користувачів та їх налаштувань

На сторінці списку користувачів є змога відсортувати їх по ролям, оновити дані користувача або позначити його як не дійсного. Для того щоб відсортувати користувачів по ролям є декілька вкладок. Також, для видалення користувача, є чекбокс з маркуванням “Old” завдяки якому можна позначити користувача, як видаленного. Список користувачів з відповідними функціями можна бачити на рисунку 3.1.

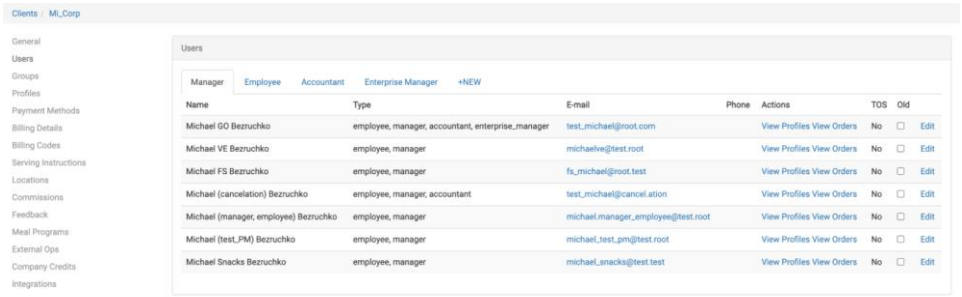

Рисунок 3.1 - Список користувачів на адміністративній панелі

Для зручної зміни користувальницької інформації, необхідний UI інтерфейс. Для змоги зміни будь якої інформації про користувача, створене модальне вікно в якому є відповідні поля для внесення оновленої інформації. Як виглядає модальне вікно зміни інформації користувача можна побачити на рисунку 3.2. Такий UI інтерфейс дозволяє вносити зміни до вже існуючих записів з користувачами.

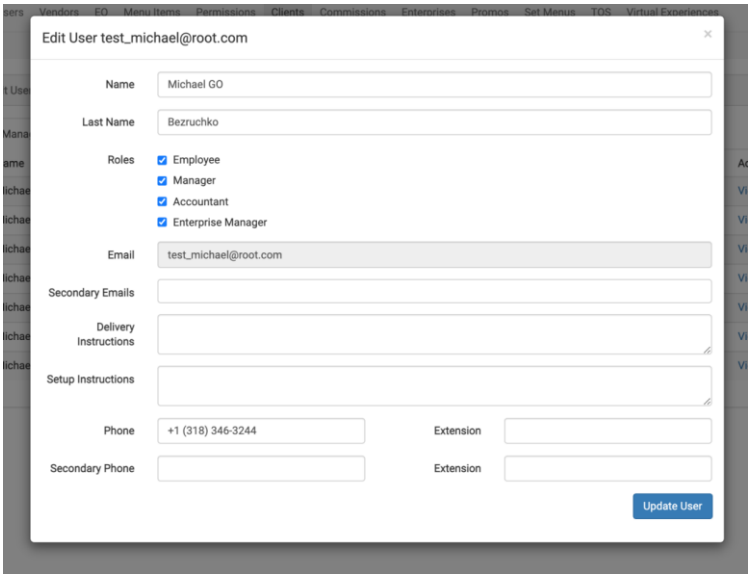

Рисунок 3.2 - Модальне вікно зміни інформації користувача

3.2 Адміністративна панель для управління профайлами

Для контролю над налаштуваннями замовлень, необхідно мати інтерфейс взаємодії з ними. Список налаштувань можна бачити на рисунку 3.3.

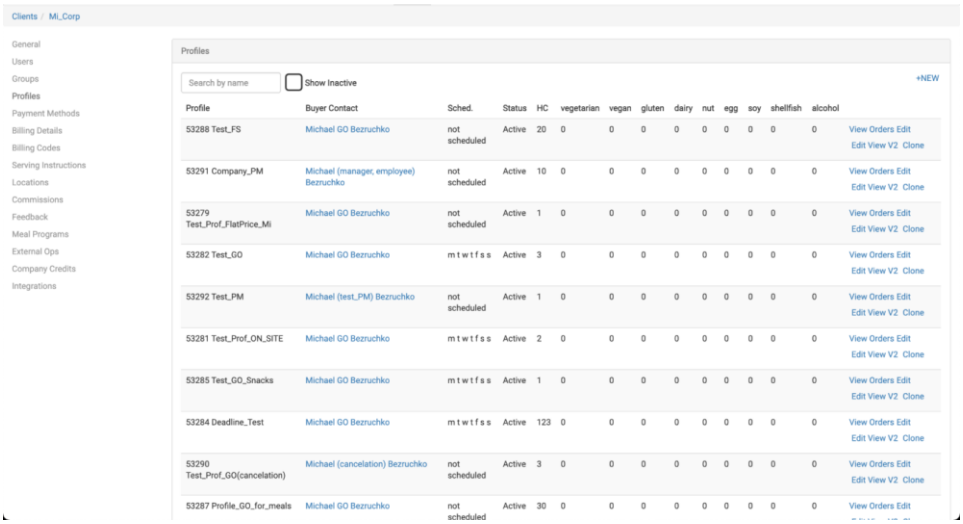

Рисунок 3.3 - Список існуючих профайлів

Для зміни налаштувань замовлень, створюється відповідний інтерфейс у адміністративній панелі (рисунок 3.4).

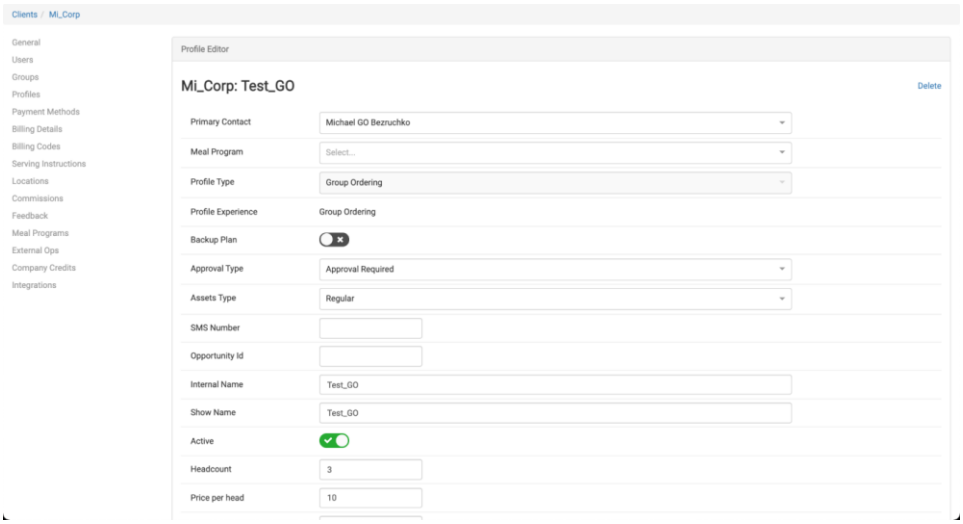

Рисунок 3.4 - Редактор налаштувань профайлу

Завдяки цьому редактору профайлів, адміністратор матиме змогу вносити зміни до налаштувать профайлу, змінювати його тип, визначити алергени, або навіть створити розклад з замовлень вперед на декілька місяців.

3.3 Створення замовлення з вибором постачальника та меню

Щоб була змога створювати нові замовлення та прикріпиляти відповідні налаштування до них, а також вказати дату, час та визначати кількість людей на яких розраховане майбутнє замовлення, було розроблено модальне вікно (рисунок 3.5).

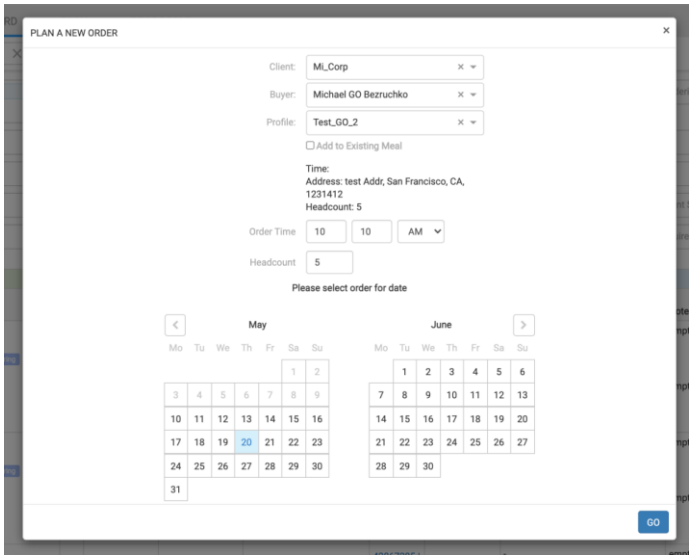

Рисунок 3.5 - Модальне вікно створення замовлення

Щоб обрати необхідного постачальника з конкретним, задовольняючим вимогам меню, був створений список постачальників з виводом відповідних меню. За допомогою дошки додавання меню, з'являється можливість відсортувати постачальників за типом, ціновою категорією та додати меню від вподобаного (рисунок 3.6).

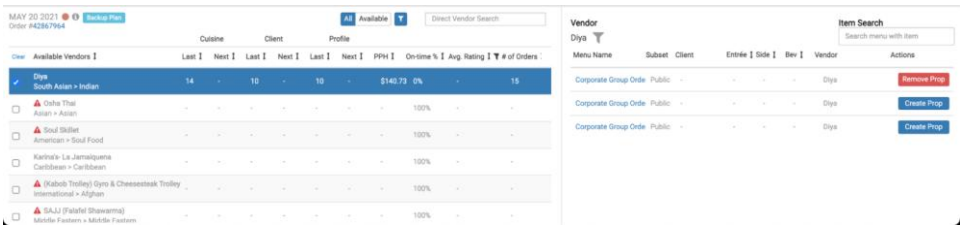

Рисунок 3.6 - Дошка додавання меню до замовлення

3.4 Управління замовленями

Для відстеження замовлень, користувач повинен мати UI інтерфейс. Панель з календарем необхідна для зручного стеження користувачем, за

своїми замовленнями. Таку панель з календарем можна побачити на рисунку 3.7.

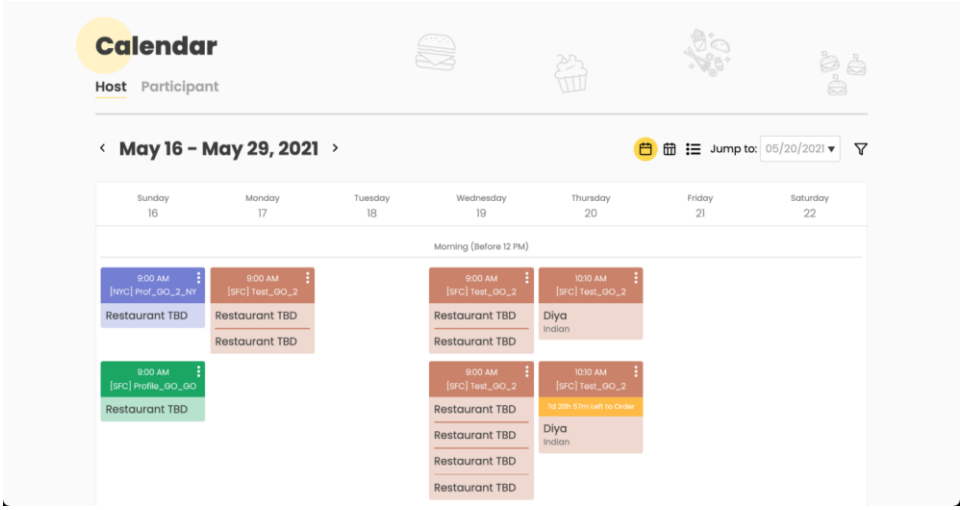

Рисунок 3.7 - Календар замовлень

Сторінка загальної інформації про замовлення, на якій можна побачити попередні підрахунки показана на рисунку 3.8. На цій сторінці є присутньою інформація про податки, чайові, та загальну вартість замовлення.

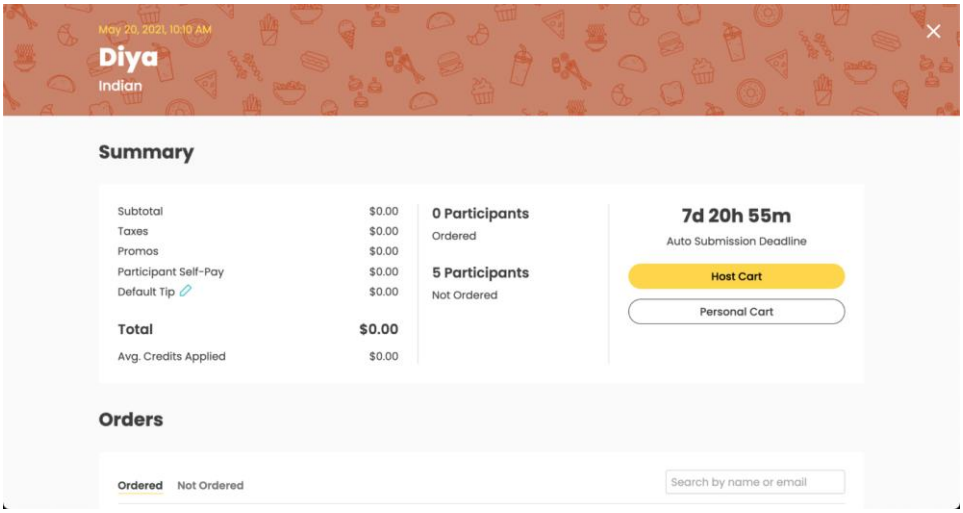

Рисунок 3.8 - Відкрите замовлення

Для вибору та підтвердження замовлення існує окрема сторінка з варіантами вибору страв за встановленим на етапі створення замовлення меню. На даній сторінці є змога додати страви, що входять до меню від постачальника, до свого замовлення. Сторінка, також, одразу містить інформацію про вартість замовлення для вже вибраних страв та суперечить чи ні будь-яка страва вподобанням користувача. Рисунок 3.9 відображає таку сторінку.

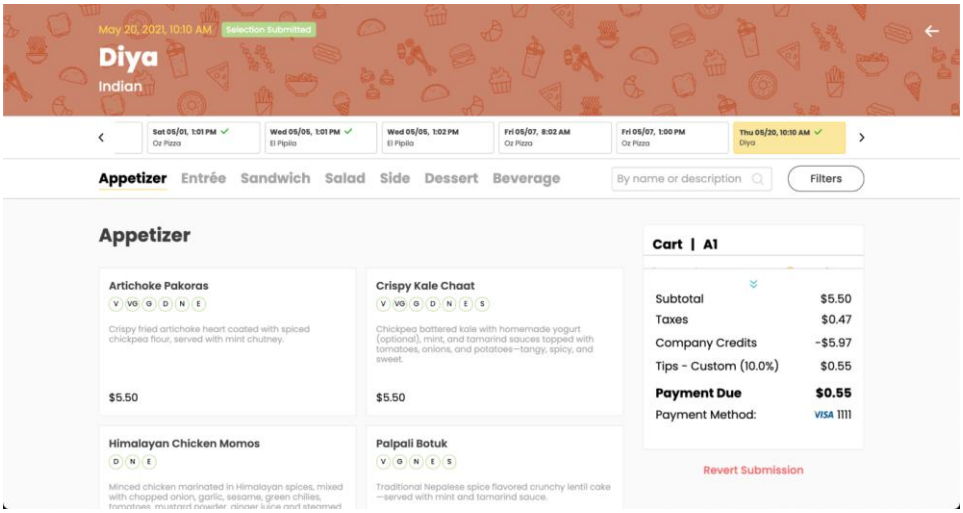

Рисунок 3.9 - Сторінка вибору страв з затвердженим меню

У випадку, якщо є необхідність дізнатись більш детальну інформацію про конкретну страву, обрати їх кількість, або ж просто додати примітку до неї при замовленні, створене модальне вікно, яке зображено на рисунку 3.10.

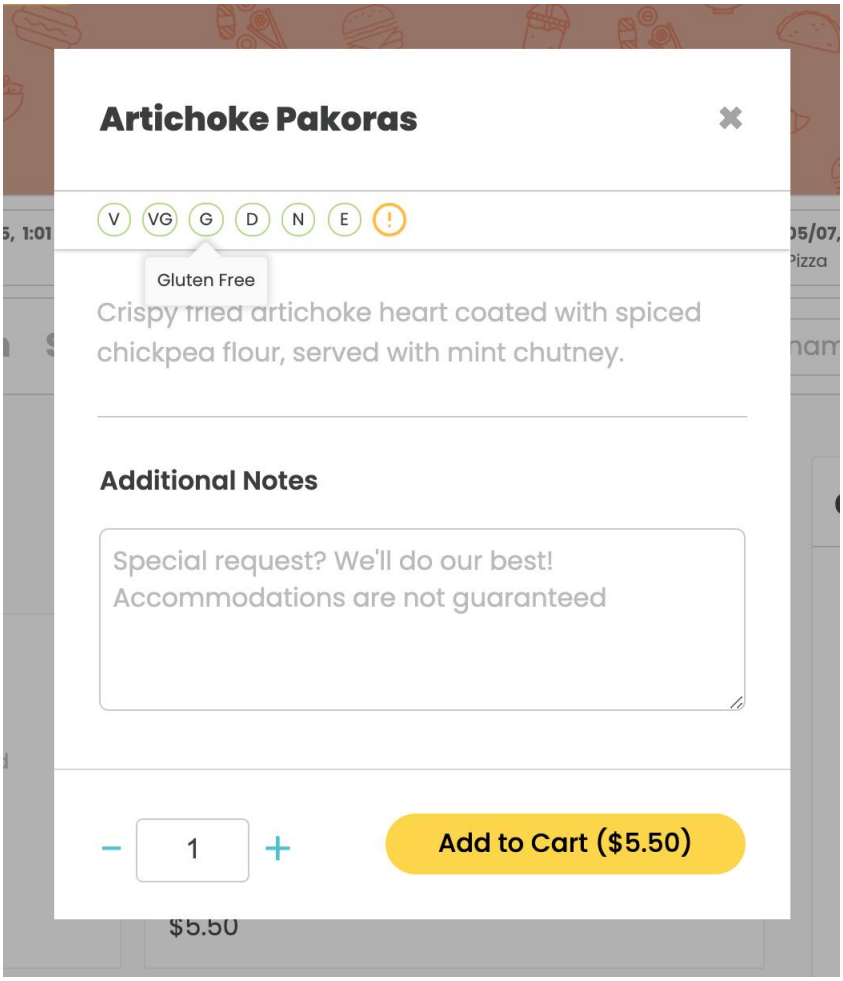

Рисунок 3.10 - Інформація про конкретну страву

3.5 Система налаштування уподобань

Для налаштування уподобань, кожному з користувачів доступна сторінка таких налаштувань у власному кабінеті користувача. Задля зміни уподобань потрібно поставити відмітку навпроти відповідного поля. Сторінку налаштувань можна бачити на рисунках 3.11 та 3.12.

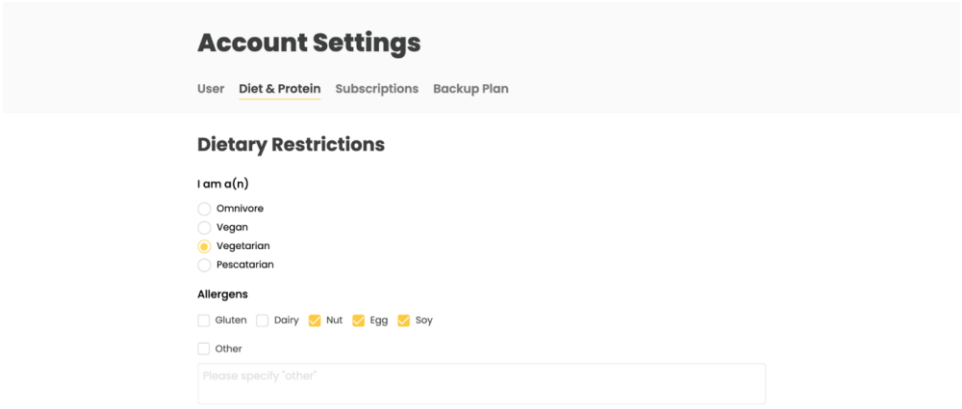

Рисунок 3.11 - Панель налаштувань алергенів

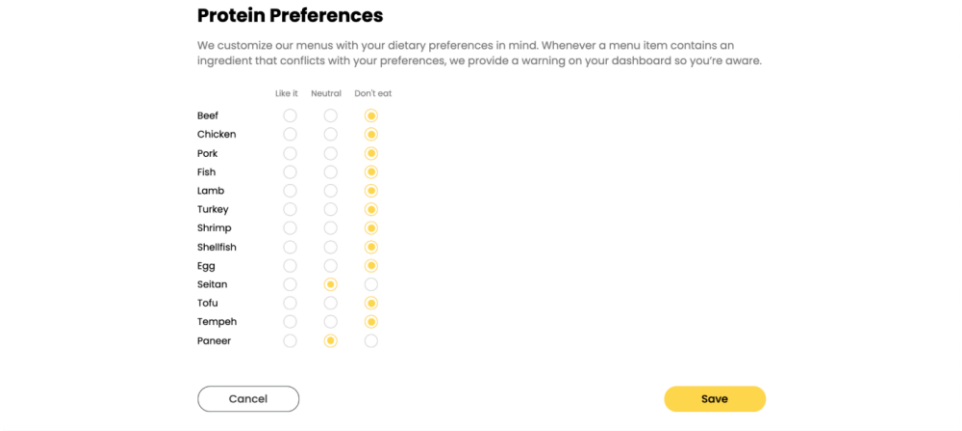

Рисунок 3.12 - Панель налаштувань вподобань

ВИСНОВКИ

В роботі проведено аналіз перспективи та проблеми кейтерингу на сьогоднішній день. Були розглянуті підходи та технології розробки програмного забезпечення, за допомогою яких і була реалізована система сервісу доставки замовлень індивідуальних та корпоративних споживачів з можливістю обирати уподобання та алергени до страв, завдяки чому максимально зменшити негативний досвід громадського харчування.

Фактично, створена програма надає можливість вирішити проблеми громадського харчування шляхом автоматизації процесу створення замовлення та підбору відповідних до уподобань клієнта страв, що вирішує проблему марнування часу на пошук та сортування страв, а також надає можливість більше часу проводити за основним родом занять.